MITRATECH

TeamConnect® Enterprise 5.0

© 2017 Mitratech

Customization Guide

TeamConnect® Enterprise 5.0 Customization Guide

Document ID: tce_5_0_cg_user_1, published on 10/11/2017

Copyright © 2017, Mitratech Holdings, Inc. All rights reserved.

Disclaimer of Warranty

Mitratech Holdings, Inc. (Mitratech) makes no representations or warranties, either expressed or implied, by or with respect to anything in this document, and shall not be liable for any implied warranties of merchantability or fitness for a particular purpose or for any indirect, special or consequential damages.

Mitratech reserves the right to not support non-standard or non-default functionality and extended functionality available in third-party software, unless specifically documented as supported or certified in the Mitratech product documentation. For further information regarding third-party non-standard or non-default functionality, please contact Mitratech Support.

This document, along with the software that it describes, is furnished under license and may be used or copied only in accordance with the terms of such license. The content of this document is furnished for informational use only, is subject to change without notice, and should not be construed as commitment by Mitratech.

The following document is for the TeamConnect™ Enterprise 5.0 release only. Though every effort was made to ensure that the information in this document is correct and reliable, Mitratech does not assume any liability for any errors encountered in this document.

If you need support for TeamConnect™ Enterprise 5.0, please contact the Mitratech support team by sending an email to: support@mitratech.com. For more information about Mitratech, visit our web site: http://www.mitratech.com.

"Mitratech", TeamConnect™ Enterprise, TeamConnect™ Legal, TeamConnect™ Legal Matter Management, Collaborati®, TeamConnect™ Collaborati Spend Management®, TeamConnect™ Deadlines, TeamConnect™ AP Link, TeamConnect™ Office Suite, TeamConnect™ Legal Reports, and TeamConnect™ SOP Manager are trademarks and products of Mitratech Holdings, Inc. All other products or services mentioned in this book are the trademarks or service marks of their respective companies or organizations.

GOVERNMENT RIGHTS LEGEND:
Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the applicable Mitratech license agreement and as provided in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013 (Feb 2012)FAR 12.212(a) (1995), FAR 52.227-19 (Dec 2007), or FAR 52.227-14, as applicable.

CONTACT US:
Mitratech Holdings, Inc.
5001 Plaza on the Lake Suite 111, Austin, TX 78746
Phone: (512) 382-7322

NOTE: Throughout Mitratech product publications, in addition to using full product names where necessary, we also use familiar and shorter terms to increase your ease of reading. You may find the following aliases for our product names:

TeamConnect for TeamConnect Enterprise
Matter Management for TeamConnect Legal Matter Management
TeamConnect Legal for TeamConnect Legal Matter Management
CSM for TeamConnect Collaborati Spend Management
Collaborati Spend Management for TeamConnect Collaborati Spend Management
SOP or SOP Manager for TeamConnect SOP Manager
Legal Hold for TeamConnect Legal Hold
Legal Reports for TeamConnect Legal Reports
Deadlines for TeamConnect Deadlines
AP Link for TeamConnect AP Link
Office Suite for TeamConnect Office Suite

Acknowledgements

This product includes software developed by the following organizations:

Apache Software Foundation (http://www.apache.org/)

OpenSymphony Group (http://www.opensymphony.com/).

The license agreements for these and other supplemental software packages can be found in your installation media in subfolder Supplemental_Software_Licenses. That subfolder also contains Open Source Components.pdf, which lists the locations, license types, and specific versions of components that are available on the web.

Table of Contents

Part I TeamConnect Setup and Development 25

1 Enterprise Customization Help 25

Introduction to Customization 26
- The Meaning of Design 26
- Preparing User and Group Accounts 26
- TeamConnect Designer User Interface 27
  - User Interface Overview 27
  - Designer Menu Bar 28
    - Backward Compatibility Settings 30
- Customization Sequence 32
  - Creating Custom Messages 35
  - Designing for Data Warehouse and Microsoft SQL Server 36

Working with System Settings 37
- Accessing System Settings 37
  - Rule Execution Properties 37
  - Miscellaneous Settings 38
- Setting up Invoices and Line Items 40
  - Defining Non-US Tax Categories 40
  - Setting up Invoice Duplicate Checking 41
  - Setting up Invoice Line Items 41
    - Setting up General Invoice Line Items Behavior 41
    - Setting up Invoice Line Items Adjustment Behavior 43
- Using Duplicate Contact Manager 46
  - Using Contact Sweeper 47
    - Prerequisites 47
    - Generate Report (Step 1 of 5) 48
    - Edit the Report (Step 2 of 5) 50
    - Upload Report (Step 3 of 5) 52
    - Fix References (Step 4 of 5) 52
    - What Kinds of References are Fixed? 53
    - Logging During the Fix References Process 54
    - Delete Duplicates (Step 5 of 5) 54
  - Preventing Duplicate Contacts 55
    - Activating Duplicate Contact Checking 55
    - Example 1 of Duplicate Contact Checking 63
    - Example 2 of Duplicate Contact Checking 65
- Using XML Worksheet Tool 68
- Creating Custom Tools 69
  - Uploading Custom Tool Files 70
  - Configuring Custom Tools 70
- Appenders 71
  - Default System and Audit Appenders 72
  - Viewing Logs Generated by File Appenders 72
  - Clearing Logs Generated by File Appenders 73
  - Rolling Over Logs Generated by File Appenders 74
  - File Appenders 74

File Appender Layouts ... 77
Pattern Layout ... 77
HTML Layout ... 80
XML Layout ... 80
SMTP Appenders ... 81
Socket Appenders ... 82
Defining System Appenders ... 84
Defining Audit Appenders ... 85
Loggers ... 87
System Loggers ... 87
Audit Loggers ... 90
About Objects ... 91
System Objects ... 91
Custom Objects ... 93
Unique Custom Object Types ... 93
Related Objects ... 93
Relationship Types ... 94
Parent-Child Relations between Custom Objects ... 96
Contact-Centric Custom Objects ... 97
Sub-Objects ... 99
Embedded Custom Objects ... 100
Creating and Defining Objects ... 101
About Object Definitions ... 102
Viewing Object Definitions ... 105
Configuring System Objects ... 108
General System Object Information ... 108
Default Object Icons ... 109
Creating Custom Objects ... 110
General Custom Object Information ... 112
Creating Custom Objects ... 115
Unique Identifiers for Custom Object Records ... 117
Creating Auto Numbering Patterns ... 119
Project Link Display Format ... 121
Creating Unique Identifier Patterns from Object Attributes ... 121
Naming Patterns for Custom Object Records ... 122
Adding Object Attributes ... 125
Attribute Format Fields ... 126
Adding Literals ... 128
Creating Auto Naming Patterns ... 130
Creating Phases in Custom Objects ... 130
Adding Phases ... 132
Editing Phases ... 132
Deleting Phases ... 133
Defining Phase Transitions for Custom Objects ... 134
Creating Phase Transitions ... 137
Editing Phase Transitions ... 137
Deleting Phase Transitions ... 138
Defining Assignee Roles for Custom Objects ... 139
Adding Assignee Roles ... 139
Editing Assignee Roles ... 141
Deleting Assignee Roles ... 142
Customizing Notifications for Object Definitions ... 143
List Displays in Embedded Objects ... 143
Displaying Embedded Records ... 144

Creating Embedded Objects ... 145
Contact-Centric Embedded Objects ... 148
Defining Display for Embedded Object Records ... 148
Converting Embedded Objects to Child Objects ... 149
Deleting a Custom Object Definition ... 152
Using Categories and Lookup Tables ... 152
Using Categories ... 153
Viewing Categories ... 153
Locations of Categories in End-User Interface ... 156
Category Tree Positions ... 156
System Object Categories ... 158
Custom Object Categories ... 158
Adding Object Categories ... 159
Editing Object Categories ... 159
Inactivating Object Categories ... 160
Activating Object Categories ... 162
Deleting Object Categories ... 162
Using Lookup Tables ... 163
Types of Lookup Tables ... 163
System Lookup Tables ... 164
Viewing System Lookup Tables ... 165
Lookup Table Structure ... 167
Contact Address Lookup Tables ... 168
Adding Lookup Table Items ... 169
Editing System Lookup Tables ... 170
Inactivating Lookup Table Items ... 170
Activating Lookup Table Items ... 172
Deleting Lookup Table Items ... 173
Custom Lookup Tables ... 173
Viewing Custom Lookup Tables ... 174
Adding Custom Lookup Tables ... 175
Deleting Custom Lookup Tables ... 176
Adding Items to Custom Lookup Tables ... 176
Editing Custom Lookup Table Items ... 177
Inactivating Custom Lookup Table Items ... 177
Activating Custom Lookup Table Items ... 179
Deleting Custom Lookup Table Items ... 180
Creating Custom Pages ... 180
Custom and System Views ... 181
Process Overview ... 182
Custom Fields ... 182
Where To Use Custom Fields ... 183
Custom Fields Tab in Object Definitions ... 185
Custom Field Name Requirements ... 187
Data Warehouse Requirements for Custom Fields ... 187
Required Fields ... 190
Field Types ... 191
List Field Type ... 194
Involved Field Type ... 194
Custom Object Field Type ... 195
Search Views for Custom Object Fields ... 198
Qualifiers for Custom Object Fields ... 199
Qualifier Syntax and Data Types ... 201
Creating Qualifiers Using Object Navigator ... 204

Creating Custom Fields ... 208
Blocks ... 210
Creating Blocks ... 211
Creating Block XML Files ... 214
Block Tags ... 215
Tags to Embed Reports in Blocks ... 220
XML File Sample ... 223
Uploading Custom Block Files ... 226
Defining Blocks ... 226
Troubleshooting Custom Blocks ... 230
Object Views ... 231
Defining Object Views ... 235
Creating Object Views ... 236
Assigning Object Views ... 238
Troubleshooting Object Views ... 238
Clearing Custom Blocks Cache ... 239
Customizing Invoice Line Item Views ... 240
Creating Search Views ... 241
Designing Search Views ... 242
Accessing the Search View Screen ... 243
Creating Search Views ... 246
Creating Search Views for Home Pages ... 247
Custom Hover Popups ... 249
Specifying Search Views Used by Global Search ... 250
Creating Searches for Approvals, Requests, and Workflow Processes ... 253
Search View Requirements for Full-Text Searching ... 254
Adding Document Content Searching to Search Views ... 254
Adding Field Content Searching to Search Views ... 255
Defining General Search View Information ... 256
Subscribing and Unsubscribing for Collections ... 260
About Auto Search ... 261
About Search Views in Search Modules ... 262
Enabling IMAP Search for Custom Object Search Views ... 263
Projects and Third-party Calendars ... 265
Custom Search Templates ... 266
Search View Results Display ... 266
Displayed Records Section Examples ... 270
Defining Object Links ... 271
Creating Object Links ... 272
Creating Results Display Columns ... 272
Search View Filter Display ... 273
Visible and Invisible Qualifiers ... 273
Qualifier Conditions ... 274
Filter Tab Example ... 277
Search Module Actions in Contact Filter Display ... 278
Creating Search View Qualifiers ... 279
Creating Section Titles ... 280
Tips for Working with Qualifiers ... 281
Troubleshooting Search Views ... 284
Creating Home Pages and Portal Panes ... 284
Home Pages and Portal Panes Basics ... 285
Portal Panes ... 287
Default Portal Panes ... 287
Opening Portal Panes ... 288

Portal Pane Settings ... 289
Portal Pane Contents ... 292
Adding Action Links ... 296
Adding Object Links ... 297
Adding Search Views ... 301
Adding Text ... 302
Adding Web URLs ... 304
Adding Custom Content ... 305
Adding WebIntelligence URLs ... 306
Portal Panes with Multiple Contents ... 307
Creating Portal Panes ... 308
Home Pages ... 309
Home Page Settings ... 310
Home Page Content ... 311
Creating Home Pages ... 312
Customization Guidelines ... 314
About Master Home Pages and Portal Pane Templates ... 315
Modifying Home Pages and Portal Panes ... 316
Synchronizing Master Pages, Templates, and Copies ... 317
About Modifying Home Pages ... 317
About Modifying Portal Panes ... 318
Modifying Portal Panes ... 320
Deleting Portal Panes ... 321
Modifying Master Home Pages ... 323
Adding Portal Panes to Home Pages ... 325
Creating Portal Panes from a Master Home Page ... 326
Using Global Navigation ... 327
Working with Tab Bar Appearance ... 328
Working with Tab Bar Contents ... 329
Using Templates ... 330
Planning Templates ... 331
Using Template Features ... 333
General Template Information ... 333
Defining Document Folders for Templates ... 334
Records Tab Structure ... 335
Creating Templates ... 336
Defining Values in Templates ... 338
Adding Fields to Templates ... 338
Field Value Types ... 341
Text or Memo-Text Fields ... 341
Number Fields ... 341
Defining Fields with Literal Values ... 342
Defining Fields with Attribute Values ... 343
Defining Fields with Formula Values ... 343
Adding Sub-objects to Templates ... 344
Adding Categories to Templates ... 345
Adding Assignees and Attendees to Templates ... 346
Deleting Sub-objects from Templates ... 348
Adding Related Objects to Templates ... 348
Creating Wizards ... 351
Designing Wizards ... 351
Writing a Wizard Specification Document ... 353
Whether to Use Templates ... 355
Designing Wizard Layout ... 356

Identifying Wizard Flow and Page Transitions ... 357
Creating Wizards ... 358
Opening the Wizard Screen ... 359
Defining General Wizard Information ... 360
Adding Pages to Wizards ... 362
Defining Page Components in Wizards ... 364
Defining Wizard Parameters ... 369
Using Actions in Wizards ... 372
Applying Automated Actions ... 373
Specifying Initial Actions ... 374
Defining Simple Actions ... 375
Defining Page Transition Logic ... 382
Creating Page Transition Rules ... 385
Checking Page Order in Wizards ... 389
Testing and Troubleshooting Wizards ... 390
Execution and Population Order ... 395
Using Rules ... 395
Rule Components ... 396
Rule Types ... 396
Security Rules ... 397
Pre-population Rules ... 398
Validation Rules ... 398
Approval Rules ... 399
Custom Action Rules ... 400
Scheduled Action Rules ... 401
User Invoked Actions ... 401
Audit Rules ... 402
Updates, Triggers, and Audit Rules ... 403
Audit Default Description ... 404
Configuring History Object Definition for Audit Tracking ... 405
Post Commit Rules ... 406
Rule Triggers ... 406
Triggering Events ... 406
Creating Rules ... 410
Defining General Rule Information ... 410
Accessing the Rules Screen ... 410
Setting General Rule Information ... 414
Rule Qualifiers ... 417
Qualifier Tab Information ... 418
Qualifier Items ... 419
Available Operators in Qualifier Items ... 421
Right Argument Options in Qualifier Items ... 423
Qualifier Logic ... 425
Creating Rule Qualifiers ... 427
Qualifier Item Examples ... 428
Limitations for User Interface Qualifiers ... 431
Rule Actions ... 432
Defining Messages for Security and Validation Rules ... 433
Defining Actions for Pre-population Rules ... 433
Defining Actions for Approval Rules ... 434
Specifying Actions for Custom Action Rules ... 442
Defining Settings for Scheduled Action Rules ... 445
Defining Actions for Audit Rules ... 448
Rule Administration ... 453

Defining User Interface Rules ... 453
Deactivating Rules ... 454
Modifying Rules ... 455
Deleting Rules ... 455
Troubleshooting Rules ... 456
Creating Routes ... 457
Accessing Routes Screen ... 458
Defining General Route Information ... 459
Route Stops ... 459
Adding Stop Members ... 463
Adding Groups as Stop Members ... 464
Adding User Paths as Stop Members ... 465
Adding Users With Roles as Stop Members ... 466
Customizing Route Stops ... 469
Adding Email Notification Recipients ... 471
Requirements for Rule Specifications ... 473
Conditional Expressions ... 475
Creating a Condition ... 476
Defining Expressions ... 478
Qualifier Logic ... 484
Finding Routes that Use a Condition ... 484
Adding Custom Code to TeamConnect ... 485
Uploading Rule Component Files ... 485
Defining Start Up Classes ... 487
Defining Rules that Use Custom Code in the User Interface ... 488
Defining Wizard Page Transition Rules ... 490
Defining Wizard Page Actions ... 491
Localizing TeamConnect ... 491
How are i18n Keys Generated? ... 492
Features that Require Your Attention ... 493
Reports ... 493
Custom Messages ... 494
Custom Blocks ... 494
Portal Panes ... 494
Rules ... 494
Wizards ... 495
Saved Searches ... 495
Upgrade Considerations for Localizing ... 495
Custom Blocks ... 495
Parameters ... 496
Date Formatting ... 496
Best Practices ... 496
Migrating Custom Designs ... 497
Requirements ... 497
Working with the Design Snapshot Tool ... 497
Creating a Design Snapshot ... 498
Creating a Design Upgrade File ... 499
Reverting to a Previous Snapshot ... 499
Working with the Design Import Tool ... 499
Performing a Design Upgrade ... 500
Working with Configuration Transfer Utility ... 501
Export Design Changes ... 502
View Older Packages ... 504
Custom Search ... 506

System Settings Excluded From Export ... 507
Import Design Changes ... 508
Imported Change History ... 510
Troubleshooting and Best Practices ... 511
Artifacts Captured for Custom Designs ... 513
Troubleshooting ... 516
Warnings ... 516
Design Import Restrictions ... 517
Source Design With No Design Component ... 517
General Troubleshooting Steps ... 518
Logging ... 518
Sample Error Messages and Tips ... 518
Error Messages and Performance Issues ... 520
BUILD FAILED message ... 520
DMT script errors ... 520
About Document Generator ... 520
Document Generator Basics ... 521
Creating Templates: Process Outline ... 525
Using Document Template Headers ... 526
Defining Filter Pages ... 526
Filters for Related Objects ... 529
Filters for Sub-objects and Single Records ... 530
Filters for Non-Related Objects ... 533
Main Filters and Sub-Filters ... 534
Defining Pages for User Input ... 538
Creating User Input Fields ... 538
Displaying Multiple Entry Fields in One Input Page ... 540
Using Document Generator Templates ... 541
Writing Document Template Text ... 542
Document Generator Tags ... 544
Document Generator Tags Summary ... 544
tc:data ... 547
tc:detail ... 552
tc:loop ... 554
Retrieving Data from All Instances of Sub-object ... 555
Retrieving Data with tc:loop Using Qualifiers ... 556
Qualifier Tag Attribute Syntax ... 557
tc:loop Qualifier Examples ... 559
tc:date ... 560
tc:user ... 560
tc:file ... 561
tc:conditional ... 562
Using tc:conditional to Compare Custom Field Values ... 565
Using tc:conditional to Compare Sub-objects ... 566
Using tc:conditional to Test for Object Attribute Values ... 566
Testing for a Category ... 567
tc:block ... 567
tc:compute ... 568
tc:operand ... 569
tc:search ... 570
tc:filter ... 572
tc:input ... 575
Nesting to Navigate ... 576
Locating Object Attribute Names ... 579

Inserting Tags and Preparing RTF Files ... 580
Completing the Document Template ... 584
Creating Document Tags ... 584
Creating Header and Converting Template to XML ... 585
Uploading Document Templates ... 587
Testing Document Generator Templates ... 587
Using EasyDocs ... 588
EasyDocs vs. Document Generator: Functional Scope ... 589
Creating a Document Template: 5-Step Process Overview ... 590
Required TeamConnect User Group Rights ... 592
Step 1: Create RTF with Merge Fields ... 592
Basic Merge Fields ... 597
Automatically Mapped System Fields ... 599
Predefined Merge Field Codes ... 599
Naming Merge Fields for Auto-mapping System Fields ... 601
Automatically Mapped Custom Fields ... 602
loop@ Merge Fields for Selecting Related or Sub-objects ... 603
When NOT to Use loop@ ... 605
Inserting loop@ Merge Fields ... 605
filter@ Merge Fields for User-selected Related or Sub-objects ... 606
When NOT to Use filter@ ... 606
Inserting filter@ Merge Fields ... 606
if@ Merge Fields for Conditional Text ... 607
Opening the Document Templates Folder ... 608
Step 2: Upload RTF File to TeamConnect ... 612
Step 3: Map RTF Merge Fields to TeamConnect Data ... 613
Opening an RTF in the Data Mapping Tool ... 613
System Field Mapping ... 614
Custom Field Mapping ... 617
loop@ Merge Field Mapping ... 618
Mapping loop@ Merge Field to Related Object or Sub-object ... 618
Mapping Merge Fields Nested within loop@ ... 619
filter@ Merge Field Mapping ... 621
Mapping filter@ Merge Field to a Related or Sub-object ... 622
Mapping Merge Fields Nested within filter@ ... 623
if@ Merge Field Mapping ... 625
Mapping to Current Date ... 628
Mapping to Current User ... 628
Mapping to a File ... 629
Converting RTF File to XML File ... 630
Mapping to XML File ... 630
Defining a User Input Screen ... 631
Step 4: Convert RTF File to a Document Generator Template ... 631
Step 5: Test Your Template ... 632
Modifying and Deleting Template Files ... 633
Modifying an Existing RTF File ... 633
Modifying Existing Data Mapping ... 634
Deleting Template Files ... 635
Troubleshooting Document Templates ... 635
Document Template Samples ... 641
Nesting to Navigate ... 641
Retrieving Data through Contact-Centric Fields ... 643
Automatically Retrieving Data from Related Objects ... 645
Retrieving Data through Custom Fields of Type Custom Object ... 646

Retrieving Data through Custom Fields of Type Involved ... 647
Retrieving Data from Child Objects ... 647
Using Qualifiers to Filter Sub-objects ... 650
Using tc:conditional ... 651
Comparing Custom Fields ... 651
Comparing Sub-objects ... 652
Testing for Existing Data ... 652
Including Information When Certain Conditions Are Met ... 653
Using Tag Attributes ... 654
Field Label Tag Attributes ... 655
Tag Attributes for All Field Types ... 656
Number Field Tag Attributes ... 658
Text Field Tag Attributes ... 660
Memo Text Field Tag Attributes ... 661
Date Field Tag Attributes ... 662
List Field Tag Attributes ... 662
Custom Object Field Tag Attributes ... 664
Involved Field Tag Attributes ... 664
Using Object Navigator ... 665
Attributes ... 666
Paths ... 668
Object Navigator Window ... 668
Starting Tables ... 671
Filtering Sub-objects and Related Objects ... 673
Creating Paths ... 676
Tips for Navigating Objects ... 678
Object Model: Read This First ... 681
Objects and Object Model ... 682
Legacy Names and Newer Names ... 682
Intended Audience ... 682
Object Model Conventions ... 683
Enumerations ... 683
Object Model Terms ... 684
Descriptions of Columns in Tables ... 685
Object Model Properties ... 687
Object Table Prefixes ... 687
Unique Codes of System Objects and Lookup Tables ... 689
Viewing XCT Files ... 691
Object Names vs. Database Table Names ... 691
List Attributes for Sub-Objects and Related Objects ... 691
Object Model: Projects ... 692
TProject ... 692
JProjAssignee ... 704
LProjAssigneeType ... 706
JProjPhase ... 708
WObjdPhaseType ... 709
WObjdPhaseTransition ... 712
JProjRelation ... 713
LProjRelationType ... 714
EProjDetail ... 716
EProjUserAccess ... 717
EProjGroupAccess ... 720
TInvolved ... 722
JInvlRelation ... 729

EInvlDetail ... 730
EInvlUserAccess ... 732
EInvlGroupAccess ... 734
TMilestone ... 737
EMileDetail ... 745
EMileUserAccess ... 746
EMileGroupAccess ... 749
Object Model: Contacts ... 752
TContact ... 752
JContAddress ... 770
LCountryItem ... 772
LContAddressType ... 773
JContPhone ... 774
LContPhoneType ... 776
JContFax ... 777
LContFaxType ... 778
JContEmail ... 779
LContEmailType ... 780
JContInetAddress ... 781
LContInetAddressType ... 782
JContSkill ... 783
LContSkillType ... 784
JContDefaultRate ... 786
JContRate ... 787
JContRelation ... 789
LContRelationType ... 791
JContTerritory ... 792
LContTerritoryType ... 793
Y_Contact_Group ... 794
JDistMember ... 796
EContDetail ... 798
EContUserAccess ... 799
EContGroupAccess ... 802
Object Model: Appointments ... 804
TAppointment ... 804
LApptArealtem ... 811
JApptAttendee ... 812
JApptResource ... 814
LApptResourceType ... 815
EApptDetail ... 816
EApptUserAccess ... 818
EApptGroupAccess ... 820
Object Model: Accounts ... 823
TAccount ... 823
JTranDetail ... 839
RTransaction ... 842
EAcctDetail ... 844
EAcctUserAccess ... 845
EAcctGroupAccess ... 848
Object Model: Expenses ... 850
TExpense ... 851
EExpeDetail ... 856
EExpeUserAccess ... 858
EExpeGroupAccess ... 860

Object Model: Tasks ... **862**
TTask ... 862
LTaskActivityItem ... 870
JTaskAssignee ... 872
ETaskDetail ... 873
ETaskUserAccess ... 874
ETaskGroupAccess ... 877
Object Model: Invoices ... **879**
TInvoice ... 879
JInvcCategoryAdjustment ... 893
JInvcHeaderAdjustment ... 895
JInvcLineItem ... 898
JInvcNonUSTax ... 906
LInvcNonUSTaxType ... 907
JInvcTimekeeperAdjustment ... 909
JLitmAdjustment ... 910
EInvcDetail ... 913
ELitmDetail ... 914
ELitmGroupAccess ... 916
ELitmUserAccess ... 918
EInvcUserAccess ... 920
EInvcGroupAccess ... 923
YCurrencyInfo ... 924
Object Model: Histories ... **925**
THistory ... 926
EHistDetail ... 930
EHistUserAccess ... 933
EHistGroupAccess ... 935
Object Model: Documents ... **937**
TDocument ... 937
JDocuContent ... 945
YDocuContentType ... 945
EDocuDetail ... 947
EDocuUserAccess ... 948
EDocuGroupAccess ... 951
Object Model: Common ... **953**
JNote ... 953
UGrupMember ... 954
WObjdDetailField ... 955
WObjdProjectInfo ... 959
YDetailLookupItem ... 964
YDetailLookupTable ... 966
YGroup ... 967
YRecentlyViewedRecord ... 970
YUser ... 971
YUserSubscribedCollection ... 977
Glossary ... **978**
2 Legal Customization Help ... **996**
Legal Customization Guidelines ... **996**
File Naming Conventions ... **997**
Filename Parts ... 997
Utility Class Files ... 999
Base Classes ... 1000
Batch Display Files for Custom Java Blocks ... 1000

Object Definitions ... **1001**
Embedded Objects ... **1002**
Categories ... **1003**
Phases and Phase Transitions ... **1004**
Assignee Roles ... **1005**
Custom Fields ... **1006**
Lookup Tables ... **1007**
Custom Screens ... **1008**
Rules ... **1009**
Reports ... **1011**
Wizards ... **1012**
Custom Tools ... **1017**
Time Entry Tool ... 1018
Editing Time Entry Tool Settings ... 1018
Editing Default Time Entry Tool Fields ... 1020
Creating Billing Time Periods ... 1022
Managing Timekeepers ... 1023
Using the Time Entry Tool as Administrator ... 1025
Tool Rights ... 1027
CSM Configuration for Matter Management ... **1028**
CSM e-Billing Roles Tab Configuration ... 1029
CSM Vendors Tab Configuration ... 1029
CSM Payment Tab Configuration ... 1030
Groups and Users ... **1030**
Service of Process Manager for Matter Management ... **1034**
Adding the CT Mapping File to Matter Management ... 1035
CT Mapping Files ... 1035
Adding the CSC Mapping File to Matter Management ... 1036
CSC Mapping Files ... 1036
3 Collaborati Spend Management Customization Help ... 1038
Setup Overview ... **1038**
Prerequisites ... 1038
Overview of the Initial Setup Process ... 1040
What Happens After Setup is Complete ... 1041
Administer CSM Setup ... 1042
Viewing Invoice Validation Rule Settings ... 1042
Accessing the Invoice Validation Rule Setting Screen ... 1043
Activating the Rate Request Approval Rule ... 1043
Activating Alternative Fee Arrangements and Rules ... 1044
Updating Additional Rules for Alternative Fee Arrangements ... 1048
Creating Alternative Fee Arrangement Pages ... 1048
Uninstalling CSM ... **1050**
Restoring CSM Default Data ... 1052
Adjusting the Invoice Validation Rule Settings ... **1054**
Fee Charges Exceed Agreed Rates ... 1054
Expense Charges Exceed Acceptable Unit Costs ... 1055
Do Not Allow Multi-Matter Invoices ... 1056
Timekeeper Billed Too Much Time ... 1057
Invoice Service/Expense Charges are Too Old ... 1059
Invoice is Too Old ... 1060
Invoice Contains Duplicate Timekeeper Charge ... 1061
Invoice Contains Duplicate Expense ... 1063
Invoice Line Item Description Contains Unauthorized Keywords ... 1065
Invoice currency doesn't match vendor currency ... 1066

Frequently Asked Questions and Troubleshooting 1067
Frequently Asked Questions 1067
Troubleshooting 1067
4 Developer's Help 1068
TeamConnect API Policy 1069
The TeamConnect API 1070
Setting up Your Development Environment 1071
API Package Classes 1072
Model Classes 1072
Enumerations 1074
Service Classes 1074
Resource Services 1075
Services in Qualifiers, Actions, and Screens 1077
Deleting Dependency Records 1078
Other Service Classes 1078
Getting Started with Custom Classes 1078
Automated Qualifiers and Actions 1082
Automated Qualifiers 1083
Automated Actions 1084
Qualifier and Action Code 1085
Parameters in Qualifiers and Actions 1087
Automated Qualifier Example 1089
Custom Pages 1091
Custom Screens 1092
Custom Screen Sample 1094
Custom Tools 1098
Custom Tool Sample 1099
Scheduled Actions 1101
Adding Java Classes for Scheduled Actions 1101
Using the API with Scheduled Actions 1102
Creating New Records 1103
Object Definitions 1103
System Objects 1105
Accounts 1105
AccountService Samples 1105
Appointments 1107
AppointmentService Samples 1107
Contacts 1109
ContactService Samples 1110
Documents 1112
DocumentService Samples 1112
Expenses 1115
ExpenseService Samples 1115
Groups 1117
GroupService Samples 1117
History Entries 1119
HistoryService Samples 1119
Invoices 1120
InvoiceService Samples 1121
Tasks 1123
TaskService Samples 1123
Users 1126
UserService Samples 1126
Custom Objects 1128

Projects ... 1128
ProjectService Samples ... 1129
Involved Parties ... 1130
InvolvedService Samples ... 1130
Related Records ... 1131
Getting Related Records of Projects ... 1132
Using Custom Fields and Primary Keys to Relate Records ... 1132
Sub-Objects ... 1133
Project and Task Assignees ... 1134
Contact Sub-Objects ... 1135
Categories ... 1135
Adding or Deleting Categories in Records ... 1136
Returning Categories for a Record ... 1137
Getting and Setting Primary Categories ... 1138
Checking Records for Specific Categories ... 1138
Custom Fields ... 1138
Requirements for Getting or Setting Custom Field Values ... 1139
Getting Custom Field Values ... 1140
Setting Custom Field Values ... 1140
Lookup Table Fields ... 1140
System Lookup Table Tree Positions ... 1142
Custom Lookup Table Tree Positions ... 1144
Getting or Setting Values in List Custom Fields ... 1145
Common API Functions ... 1145
Running Code as the System User ... 1147
Getting Current User Information ... 1149
Security Rights ... 1150
Record Security ... 1150
Functional Security ... 1151
System and User Settings ... 1152
Adding Custom Settings ... 1153
Internationalization ... 1154
Searching for Records ... 1155
Creating an API Search View ... 1157
Search Criteria ... 1157
Searching on Fields ... 1161
Field Path Searching ... 1162
Search Criteria Operators ... 1164
Search Parameters ... 1164
Executing a Search ... 1165
Logging ... 1166
Specifying Logging Messages ... 1167
Checking Logger Levels ... 1168
Java Samples ... 1169
Partial Samples to Demonstrate Concepts ... 1169
Automated Qualifiers ... 1171
Automated Actions ... 1179
Rule Actions ... 1179
Parameterized Actions ... 1187
Wizard Page Actions ... 1192
Searching Samples ... 1197
5 Web Services Help ... 1200
Getting Started ... 1200
Overview ... 1200

Supported Operations 1202
Key Concepts 1206
Repository Classes 1206
Selective Record Updating and Reading 1207
Data Types 1207
Request and Response Formats 1207
Error Handling 1208
Record Unique Keys 1208
Requirements 1208
Generating Client Source-code and API 1209
Creating an Application 1213
Client Proxy 1213
SOAP Message Security Header 1214
Putting Components Together in a Client Application 1215
Common Functions 1215
Creating Records 1216
Updating Records 1216
Reading Records 1216
Searching Records 1216
Working with Categories 1218
Working with Custom Fields 1220
Deleting Records 1221
Web Service API Reference 1222
API Reference Summary 1222
AccountRepository 1223
AppointmentRepository 1225
ContactRepository 1227
DocumentRepository 1233
ExpenseRepository 1236
GroupAccountRepository 1237
HistoryRepository 1238
InvoiceRepository 1239
LookupTableSource 1243
ProjectRepository 1243
InvolvedRepository 1246
TaskRepository 1247
UserAccountRepository 1249
Repository Method Details 1250
AccountRepository Method Details 1250
insertAccount 1250
updateAccount 1251
readAccount 1251
readAccountsByCriteria 1252
readChildAccounts 1253
activateAccount 1253
deactivateAccount 1254
allocateMoney 1254
transferMoney 1254
withdrawMoney 1255
deleteAccount 1255
readRecentlyViewedAccounts 1256
AppointmentRepository Method Details 1256
insertAppointment 1256
updateAppointment 1258

readAppointment ... 1258
readAppointmentsByCriteria ... 1259
deleteAppointment ... 1259
readRecentlyViewedAppointments ... 1260
ContactRepository Method Details ... 1260
insertContact ... 1260
updateContact ... 1262
readContact ... 1262
readContactsByCriteria ... 1263
deleteContact ... 1264
ReadRecentlyViewedContacts ... 1264
DocumentRepository Method Details ... 1264
insertDocument ... 1265
createShortcut ... 1267
createSubFolder ... 1267
createHyperlinkDefaultCategory ... 1268
createHyperlink ... 1268
updateDocument ... 1269
readDocument ... 1270
readDocumentsByCriteria ... 1270
readDocumentByPath ... 1271
readChildDocuments ... 1272
readChildDocumentForName ... 1272
readDocumentFolderForDocumentOwner ... 1273
checkIn ... 1274
checkOut ... 1274
undoCheckOut ... 1275
revert ... 1275
moveDocument ... 1276
copyDocument ... 1276
deleteDocument ... 1277
ReadRecentlyViewedDocuments ... 1277
setDocumentAsRecentlyViewed ... 1277
readFolderHierarchy ... 1278
ExpenseRepository Method Details ... 1278
insertExpense ... 1279
updateExpense ... 1279
readExpense ... 1280
readExpensesByCriteria ... 1281
postExpense ... 1281
voidExpense ... 1282
deleteExpense ... 1282
readRecentlyViewedExpenses ... 1282
GroupAccountRepository Method Details ... 1283
insertGroupAccount ... 1283
updateGroupAccount ... 1284
readGroupAccount ... 1284
readGroupAccountsByCriteria ... 1285
deleteGroupAccount ... 1286
readRecentlyViewedGroupAccounts ... 1286
HistoryRepository Method Details ... 1287
insertHistory ... 1287
updateHistory ... 1288
readHistory ... 1288

readHistoriesByCriteria 1289
deleteHistory 1290
readRecentlyViewedHistories 1290
InvoiceRepository Method Details 1290
insertInvoice 1291
updateInvoice 1293
readInvoice 1293
readInvoicesByCriteria 1294
postInvoice 1295
voidInvoice 1295
deleteInvoice 1296
readRecentlyViewedInvoices 1296
adjustInvoiceHeader 1296
readActiveApprovals 1297
readCompletedApprovals 1298
readInvoiceApprovalsPendingOnPost 1298
LookupTableSource Method Details 1299
readSystemLookupTable 1299
ProjectRepository Method Details 1300
insertProject 1300
updateProject 1301
readProject 1302
readProjectsByCriteria 1302
readChildProjectsForEntityType 1303
changePhase 1304
deleteProject 1305
readRecentlyViewedProjects 1305
readProjectEntityTypes 1305
readProjectIntegrationSearches 1306
readProjectsUsingSearch 1306
InvolvedRepository Method Details 1307
insertInvolved 1307
updateInvolved 1308
readInvolved 1309
readInvolvedsForProject 1309
readInvolvedsByCriteria 1310
deleteInvolved 1311
TaskRepository Method Details 1311
insertTask 1311
updateTask 1312
readTask 1313
readTasksByCriteria 1313
postTask 1314
voidTask 1314
reassign 1315
deleteTask 1315
readRecentlyViewedTasks 1315
UserAccountRepository Method Details 1316
insertUserAccount 1316
updateUserAccount 1318
readUserAccount 1318
readUserAccountsByCriteria 1319
deleteUserAccount 1320
readRecentlyViewedUserAccounts 1320

REST API HTTP Methods ... 1321
Abstract Classes ... 1324
Category and Custom Field Data Objects ... 1326
Data Objects Used in Search Criteria ... 1327
Frequently Asked Questions ... 1329
Batch Operations ... 1329
Operations Not Supported In Web Services ... 1329
Cosmetic Data Objects ... 1330
Client Application Components (Java/Apache CXF) ... 1330
Client Proxy Sample Code ... 1331
Security Headers Sample Code ... 1332
Client Application Sample Code ... 1333
Code Samples (Java/Apache CXF) ... 1335
Contacts ... 1336
Creating Contacts ... 1336
Updating Contacts ... 1338
Reading Contacts ... 1340
Searching for Contacts ... 1340
Deleting Contacts ... 1342
Accounts ... 1343
Creating accounts ... 1343
Updating accounts ... 1344
Reading accounts ... 1345
Reading child accounts ... 1346
Searching accounts ... 1347
Activating accounts ... 1348
Deactivating accounts ... 1349
Allocating money to accounts ... 1349
Transferring money between accounts ... 1349
Withdrawing money from accounts ... 1350
Deleting accounts ... 1350
Appointments ... 1350
Creating appointments ... 1350
Updating appointments ... 1351
Reading appointments ... 1352
Searching appointments ... 1353
Deleting appointments ... 1354
Document Records ... 1354
Creating document records ... 1354
Updating documents ... 1355
Reading documents ... 1356
Reading child documents for a document ... 1357
Reading documents by path ... 1358
Reading the child document of a parent folder ... 1359
Reading the Document Folder for a User ... 1360
Searching documents ... 1361
Creating document shortcuts ... 1363
Creating subfolders ... 1363
Creating hyperlinks with the default Document category ... 1363
Creating hyperlinks ... 1364
Copying documents ... 1364
Moving documents ... 1364
Checking out documents ... 1365
Undoing a document checkout ... 1365

Checking in documents ... 1365
Reverting a document to a previous version ... 1365
Deleting documents ... 1366
Expenses ... 1366
Creating expenses ... 1366
Updating expenses ... 1366
Reading expenses ... 1367
Searching expenses ... 1368
Posting expenses ... 1369
Voiding expenses ... 1369
Deleting expenses ... 1369
Group Accounts ... 1369
Creating group accounts ... 1370
Updating group accounts ... 1370
Reading group accounts ... 1371
Searching group accounts ... 1372
Deleting group accounts ... 1372
History Records ... 1373
Creating history records ... 1373
Updating history records ... 1373
Reading history records ... 1374
Searching history records ... 1375
Deleting history records ... 1376
Invoices and Line Items ... 1376
Creating an Invoice with Line Items ... 1376
Updating and Adjusting an Invoice ... 1378
Adjusting Invoice Line Items ... 1379
Reading an Invoice ... 1380
Searching for Invoices ... 1381
Posting an Invoice ... 1383
Voiding an Invoice ... 1383
Searching for Line Items ... 1383
Deleting an Invoice ... 1385
Projects (Matters) ... 1385
Creating a Project With Assignees and a Custom Field ... 1385
Updating a Project by Adding an Embedded Object Record ... 1387
Changing a Project's Phase ... 1388
Reading Projects ... 1388
Reading Child Projects for a Record ... 1389
Searching for Projects ... 1389
Deleting a Project ... 1390
Involved Records ... 1390
Inserting Involved records ... 1390
Updating Involved records ... 1391
Reading Involved records ... 1392
Reading Involved Parties for a Project Record ... 1392
Searching Involved Party Records ... 1393
Deleting an Involved ... 1394
Tasks ... 1394
Creating tasks ... 1394
Updating tasks ... 1395
Reading tasks ... 1395
Searching tasks ... 1396
Posting tasks ... 1397

Voiding tasks ... 1397
Reassigning tasks ... 1398
Deleting tasks ... 1398
User Accounts ... 1398
Creating user accounts ... 1398
Updating user accounts ... 1399
Reading user accounts ... 1399
Searching user accounts ... 1400
Deleting user accounts ... 1401
Glossary ... 1401

Index 1403

1 TeamConnect Setup and Development

Welcome to the *TeamConnect® Setup and Development Help*.

I use TeamConnect as an...	I need help with...
IT Manager	Enterprise Setup and Design Legal Setup Collaborati Management Setup API Development Building Applications

1.1 Enterprise Customization Help

Welcome to the *TeamConnect® Enterprise Customization Help*.

System Settings	Objects
Categories	Lookup Tables
Custom Pages	Search Views
Home Pages	Portal Panes
Global Navigation	Templates
Wizards	Rules
Conditional Expressions	Automated Qualifiers
Matter Management Design	Localization
Custom Designs	Document Generator
EasyDocs	Object Navigator

1.1.1 Introduction to Customization

Developing your custom TeamConnect design involves research and planning. Business analysts gather the requirements to represent your organization's business model, and technical analysts develop the corresponding functional specifications.

Be sure to develop or obtain all of the necessary design documents before you and your team begin the customization process. Once your design is completed, you can set up and configure TeamConnect to translate the design into a fully functional application to meet your organization's needs.

Before you start implementing your design in TeamConnect, do the following operations:

- Install TeamConnect according to the requirements.
- Create your own user account and ensure that it belongs to a user group with **Setup** rights.
- Familiarize yourself with TeamConnect and its Designer user interface.
- Plan your implementation strategy according the order, process, and dependencies listed in Customization Sequence.

1.1.1.1 The Meaning of Design

As used in this document, the word "design" has several definitions:

- The abstract models and requirements that your organization develops before implementing TeamConnect.
- The collection of object model definitions, rules, templates, and other system components that results from your implementation of those abstract models and requirements in TeamConnect.
- That part of the TeamConnect database that defines the system components only, excluding data entered by end users. That is the specific definition used by Migrating Custom Designs.

Most of this document uses the second definition in the list above.

1.1.1.2 Preparing User and Group Accounts

Even though users and user groups are not administered in the Designer, it is important to have a well-developed structure of groups already in place before doing customization. Some of the tasks in customization, such as assigning object rights and designating workflow stops in routes, require that the appropriate groups already exist.

Even if you are testing with only a small number of users, the structure of your groups should reflect how TeamConnect will be used in full production. It is relatively simple to add many more users to existing groups later, in production, without requiring any further customization.

You will need your own user account if one has not already been made for you during the installation process. Your account must belong to a group that has **Setup** rights. You can also get access to Setup if your user account is designated as a "superuser". However, for realistic testing, it is preferable that your user account is designated as "normal", with appropriate group memberships and Setup rights.

Consult the person who installed TeamConnect to see if you already have such a user account. If not, have an account created for you.

As you customize TeamConnect by adding custom objects or tools, assign rights to them to the appropriate user groups, including your own user group. Otherwise, you will not be able to access and test them.

1.1.1.3 TeamConnect Designer User Interface

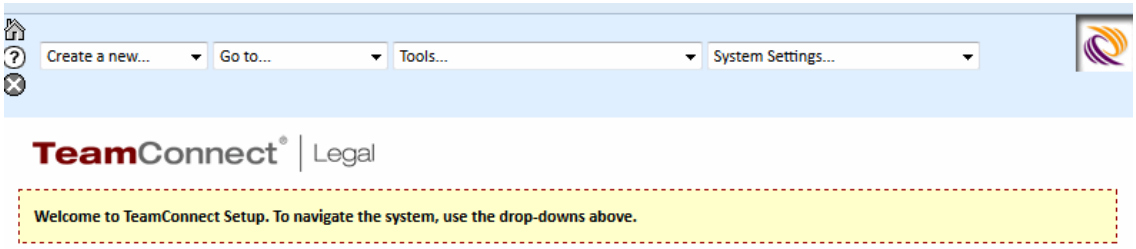
Most customization tasks do not require extensive technical skills because you perform them using the TeamConnect Designer user interface. For example, you can develop most business rules directly through the user interface.

Depending on your design, you may also need certain technical skills, such as a knowledge of Hypertext Markup Language (HTML) and Extensible Markup Language (XML), to create custom pages. Some customization of rules and pages requires expertise with Java and JavaScript.

To access the Designer user interface

1. Log in to TeamConnect with Setup rights.
2. Click the Setup link at the top of the page (upper right).

The main access page of the Designer user interface opens in a separate window, as shown in the following image.



Designer Main Page

1.1.1.3.1 User Interface Overview

Most Designer user interface elements are self-explanatory. However, some procedures require knowledge of TeamConnect itself.

For example, you may define patterns to automatically name records, such as the date on which the record was created, the record's default category, and the name of the main involved party. To create such patterns, you must identify the location of these attributes within TeamConnect's object model (see the various Object Model appendices for details). Thus you would require Object Model knowledge for this task.

The Designer user interface has the following types of screens:

- Record Screens—Most Designer configuration screens include tabs and a toolbar. For example, object definition screens include tabs where you can create and define the corresponding components, such as the **Templates** and **Wizards** tabs of the Account object.

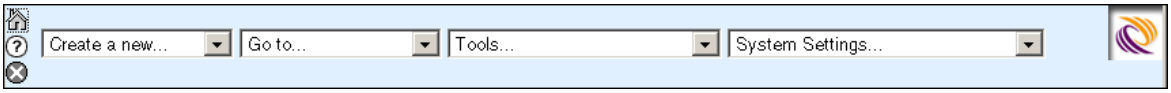
  Unlike the end-user interface, the Designer user interface does not have the Read-only display mode. Most fields in the screens are editable at all times.

- Batch Screens—Many configuration screens include batch screens, which have a row of data entry fields for entering multiple items at the same time. For example, the **Lookup Tables** screen allows you to enter the names and tree positions of dozens of lookup table items at the same time.
- Using Object Navigator—A user interface tool for defining paths to attributes in the object model. See Using Object Navigator and Object Model: Read This First provides information about the objects, attributes and relationships in TeamConnect's object model that you can use to create your custom designs. This reference points you to additional reference tables that provide more information about object models.

While developing your design, you are testing it in the end-user interface as well. You may also need to use that interface to create and define users and groups, access rights, system settings, and so on.

1.1.1.3.2 Designer Menu Bar

The Designer menu bar contains a set of drop-down lists, which provide a quick access to the appropriate records and functions.



Designer Menu Bar

Designer Menu Bar Fields

Field or control	Icon or option	Description	
Logo		TeamConnect logo. When the image in the icon is "spinning," it indicates that the system is processing a request, such as opening a record, searching for a records, or another operation.	
Home		Opens your **Home** page. For details on home pages, see Creating Home Pages and Portal Panes.	
Help		Provides access to the Designer help files.	
Sign Off		Allows you to sign off from TeamConnect..	
Create a new	**Object Definition**	Displays the **General** tab of the corresponding screen.	For details on how to define objects, see Creating and Defining Objects.
	Route		For more information on creating routes, see the Creating Routes.

	Home Page		See Home Pages.
	Portal Pane		See Portal Panes.
Go to	**Object Definitions**	Displays a list of available system and custom objects. Click the appropriate hyperlink and the corresponding object definition screen appears with its **General** screen opened by default. For details on the information displayed in this screen, see General System Object Information and General Custom Object Information.	
	Routes	Click the appropriate hyperlink and the **General** tab of the corresponding screen appears by default.	For more information on routes, see Creating Routes.
	Home Pages		See Home Pages.
	Portal Panes		See Portal Panes.
	Lookup Tables	Displays the **System** tab of the tables screen, where you can add, modify, inactivate, and delete the necessary items in the appropriate system tables. For more details, see the Types of Lookup Tables. For details on creating custom lookup tables and using them in custom fields of type List, see Using Lookup Tables.	
	Global Navigation	Click the hyperlink to show the tabbed work screen.	See Using Global Navigation.
Tools	**Administer Custom Tools**	Displays the **Administer Tools** screen where you can add or delete custom tools, and rename system tools. See Creating Custom Tools.	
	Misc. Settings	Displays the **Miscellaneous Settings** screen where you can add custom fields for integration of TeamConnect with other products. See Miscellaneous Settings.	
	Contact Sweeper	Run a tool that identifies possible duplicate contacts and allows you to choose which ones to consolidate and delete. See Using Contact Sweeper.	
	Design Snapshot	Run a tool that captures complete information about the object definitions and other TeamConnect components in the current application.	

	Design Import	Run a tool that uses information from Design Snapshot to change the current application design to match the snapshot.	
	XML Worksheet	Launch a tool that allows you to upload XML requests to the application and view the responses.	
System Settings	**Rule Execution**	Displays the corresponding tab in the **System Settings** screen, where you can configure the appropriate setting for your application.	See Accessing System Settings.
	Custom Messages	Lets you create custom messages and assign unique messages keys.	See Creating Custom Messages for more information.
	Clear Cache 2.x user interface Settings	Allows you to clear the cache, which may be required when you create custom pages. See Creating Custom Pages.	
	Default Object Views	See Object Views.	
	History	Displays a Search page where you can set your search criteria for the history entries associated with system settings.	
	Backward Compatibility Settings	See Backward Compatibility Settings.	
	System Appenders	Displays the corresponding tab in the **Logging** screen, where you can define logs and set the status of loggers.	
	Audit Appenders		

1.1.1.3.2.1 Backward Compatibility Settings

Each new version of TeamConnect contains new and updated features. The **Backward Compatibility Settings** page provides several options that let you return to previous behavior until you are ready to shift to the new features.

Important: *You can check and uncheck these boxes at any time, but it is not recommended for settings that affect how data is stored. Make the switch in a test environment to ensure that everything works properly.*

To open the Backward Compatibility Settings page

1. Open the Designer.
2. In the **System Settings** drop-down list, select **Backward Compatibility Settings**.
3. On the **Backward Compatibility Settings** page, select one or more of the options in the Backward Compatibility Settings Page table.

Note: *Most of the boxes on this page are unchecked by default. Adding a check-mark to a box indicates that you want to use the old behavior. This does not apply to the **Optimize cascading security** option, which is checked by default. Please see the details below.*

Backward Compatibility Settings Page

Control	Options
Allow Duplicate Invoices (Same invoice number, invoice date, year, and vendor)	• If checked, an error message appears when posting duplicate invoices. • If unchecked, allows duplicate invoices for the purposes of data migration of duplicate invoices for the Dynamic Fields Conversion upgrade tool and Workflow upgrade tool.
Enable wild card characters in XML layer search requests	• If checked, it is not possible to use wild cards when writing search requests in the XML layer. • If unchecked, wild cards are valid.
Always use 'ISO- 859-1' for encoding characters entered in text/memo fields	• If checked, the default is the ISO/IEC 10646 character set. • If unchecked, the eight-bit coded character set is used in text and memo fields.
Optimize Cascading security	• If unchecked, cascading security proceeds in a single batch with one commit. • If checked, if cascading security is enabled for a custom object definition, the security cascade proceeds in batches of 100 and commits after each batch. Any failures in batches is logged and do not prevent the completion of later batches.

	Note: *This box is checked by default.*
Ignore security cascading from document folders	• If checked, security propagates based on the Replace Rights on subfolders and/or Replace Rights on existing files sections in the security block of the document folder. • If unchecked, modifying the security of a document folder will not propagate the security changes to its child files and folders.
Take user to document properties page when clicking on a shared persistent URL of a document--	• If checked, clicking a document or document folder URL link in an email message opens the TeamConnect **Documents General** page for the referenced document. • If unchecked: o **For documents**—Clicking a document URL link in an email message opens the document in a browser. If the document URL links to a file that may contain other files, such as a zip file, the browser page opens the TeamConnect Documents page that contains the referenced file. o **For document folders**—Clicking a document folder URL link in an email message opens the TeamConnect **Documents** page that contains the referenced folder. ***Notes:*** *Users must be logged in to TeamConnect and have the appropriate rights to access documents, document properties, or document folders.* Browser settings determine the way that documents are displayed. For example, if the browser is set to open a new tab whenever a document is opened, clicking a persistent document URL link always opens the document in a separate tab.

1.1.1.4 Customization Sequence

Some customization tasks are sequential and others can be performed concurrently. For example, you cannot create custom fields until you have created the categories to which the fields belong.

Once you are done creating custom fields, you can create rules, define wizards, or define home pages in any order.

Creating custom objects and fully developing their pages, rules, search views, and so forth usually requires the most work. System objects come "out of the box" fully developed, although your organization's design may require some custom pages, wizards, and other extras for system objects.

Important: *Collaborati Spend Management (CSM) is delivered as a group of custom objects. Although it is possible to alter the properties of these objects, you should not do so, since the alterations could prevent future updates to CSM from installing properly.*

Customizing System Objects

Customizing system objects (optional) is typically done in the following sequence:

1. Define general object information.
2. Add items to system lookup tables.
3. Define categories.
4. Create custom lookup tables and their items.
5. Create custom fields.
6. Create custom pages.
7. Configure Rules and workflow.
8. Create Document Generator templates.
9. Test the components of your custom design.

Creating Custom Objects

You must create custom objects in the following sequence:

1. Define general object information.
2. Define unique identifiers for records.
3. Define record name patterns.
4. Define phases and their transitions.
5. Add assignee roles.
6. Add items to system lookup tables.
7. Define categories.
8. Create custom lookup tables and their items.
9. Create custom fields.
10. Once the previous tasks are completed, you can perform the following tasks in parallel or in any sequence:

- Create custom pages.
- Configure Rules and workflow.
- Create Document Generator templates.

11. Test the components of your custom design.

Creating Custom Pages

You can define the following types of custom pages:

- **Custom object views—**Layout of fields and record pages
- **Wizards—**An tool to guide end-user through creating or updating records
- **Search views—**Pages for finding and displaying lists of records
- **Home pages—**Provide a range of information and links in one location
- **Portal panes—**Components of home pages that display sets of information and links

You can create custom pages in the following sequence:

1. Define blocks.
2. Define object views.
3. (optional) Define templates for wizards.
4. Define any user groups necessary for wizard rules.
5. Define wizards.
6. Define search views.
7. Define portal panes.
8. Define home pages.

Creating Custom Object Views

Object views define the layout of fields and record pages. You can customize and assign object views for different user groups.

To successfully create object views, you should understand the following:

- Object categories, their purpose, full tree positions, how they are created, where they appear in the end-user interface, and so on. See Using Categories.
- The differences between system and custom object views. See Custom and System Views.

You can create custom object views in the following sequence:

1. Create a field specification document, including categories and display requirements for the object definition.
2. Create the custom object definition (except for system objects).
3. Create any necessary categories.

4. Create custom fields.
5. Create blocks.
6. Create object views.

XML Templates

The design of Custom Java Blocks is outside the scope of this documentation. However, if you do work with Custom Java Blocks, be aware of the XML templates available in the `\utitilities \samples\screens` subfolder of your program directory. By using these templates for your work, your CJBs will preserve the look and feel of "out of the box" TeamConnect.

1.1.1.4.1 Creating Custom Messages

For error messages that may appear as a result of custom java rules, custom java blocks, or custom tools, TeamConnect lets you create custom message keys and associated message text for localization purposes.

You can define for the following:

- Object-definition-specific custom messages used in custom rules and screens.
- Common custom messages that are used in custom tools, custom portal pane content, and custom screens.

Once defined, when you export data from TeamConnect for localization, the message keys and message text appear on the Excel spreadsheet on the **Custom Resources** tab. You can enter your translations on this tab and then import the translated text back to TeamConnect. This allows users to see custom messages in their preferred language.

Once you create a message key, you cannot edit it, but you can edit the associated message. You can delete a custom key and its associated message.

Note: *If the text of a custom message contains parameter tokens ({0}, {1}, etc.) these tokens must be retained in the translated text for each locale.*

To create custom messages

1. Open the Designer.
2. Select one of the following options:
   - **For object definition-specific custom messages—**Click the **Go to** drop-down list, and then select **Object Definitions**. Select the object definition for which you want to create custom messages. Click the **Custom Messages** tab.
   - **For common custom messages—**Click the **System Settings** drop-down list, and then select **Custom Messages**. If necessary, click the **Custom Messages** tab.
3. (optional) In the **Number of entries you would like to add** drop-down list, enter the number of keys with messages that you want to create.
4. In the **Message Key** field, enter the appropriate message key.

All common custom message keys are prefixed with *custom.common.*; all object definition message keys are prefixed with *custom.* only. You do not have to enter these prefixes or the period—they are automatically added to the front of the message key when you click **add more**.

A message key may contain up to 250 alphanumeric characters.

5. In the **Default Value** box, enter the appropriate message text. The message may contain up to 4000 characters. A Default Value message must have an associated message key.
6. Click **add more**.

To edit custom messages

Note: *You cannot edit the* ***Message Key****, but you can edit the associated* ***Default Value*** *message.*

Click the check-box for the **Default Value** that you want to edit, and then click **edit**. Make the appropriate changes and then click **ok**.

To delete custom messages

Click the check-box for the **Message Key** that you want to delete, and then click **delete**. Deleted keys are no longer available when you re-import localized date into TeamConnect.

1.1.1.4.2 Designing for Data Warehouse and Microsoft SQL Server

Before you begin changing your TeamConnect design, you must be aware of some limitations imposed by Microsoft SQL Server, particularly with regard to the Data Warehouse feature of TeamConnect.

The system objects and custom objects contained in TeamConnect can be synchronized with database tables in the Data Warehouse, a feature that is used for reporting. However, tables in Data Warehouse can sometimes have a very large number of columns, particularly when the related TeamConnect object has a large number of categories. In Microsoft SQL Server, Data Warehouse tables support a maximum of 8060 bytes per row. You should design your categories and reportable fields so that the 8060 byte limit cannot be exceeded. (For the bytes allocated per column for each field type, see Data Warehouse Requirements for Custom Fields.) These amounts are subtracted from MS SQL Server's maximum allocation of 8060 bytes per row.

For example, if there are 30 custom fields of the type Text within a single category, the Data Warehouse table has 30 columns that use 250 bytes each. Altogether, 30 custom fields of the type Text use 7500 bytes of the 8060-byte allocation, which only leaves 560 bytes for other field types in that category.

When run with Microsoft SQL Server, Data Warehouse has a 250-byte limit for custom fields of the type **Text**. If a user enters data in excess of this 250-byte limitation, the Data Warehouse scripts truncate the text when populating the table. In addition, some Non-ASCII characters use two bytes per character of the 250-byte data length limitation. To make sure that users do not exceed the 250-byte limit, which results in truncated data in reports, choose one or more of the following options:

- Use memo text fields instead of custom fields of the type **Text** if users might exceed the 250-byte limit.
- Develop rules in TeamConnect to enforce the 250-byte limit for each custom field of the type **Text**.

1.1.2 Working with System Settings

Most customization tasks involve creating or changing object definitions, search views, new custom pages, or other TeamConnect components, then modifying TeamConnect to use these new components. However, some tasks use existing TeamConnect pages and features primarily to configure or reconfigure the system's behavior settings.

You may access these configuration settings using:

- **Tools** drop-down list
- **System Settings** drop-down list
- Screens inside the related Object Definitions, which allow you to change the behavior of specific system objects

As necessary, you may also create and upload custom tools to help you with setting these configurations.

All of these configuration or reconfiguration features are available to you in TeamConnect.

1.1.2.1 Accessing System Settings

Many settings and preferences that control the behavior of TeamConnect are accessed through the **Admin** tab in the user interface. However, some settings and preferences are accessed through Designer.

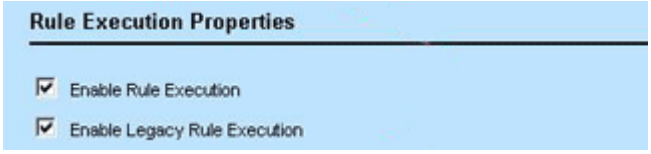
In addition to the TeamConnect system settings and predefined tools, your organization may require more configuration options to be set. This documentation provides only the following information:

- How to configure custom tools that you may develop or purchase, such as Time Entry, Batch Invoicing, or Custom Fields for Invoice Line Items tool. See Configuring Custom Tools.
- How to create custom fields, such as password fields, for integrating your TeamConnect application with other systems. See Miscellaneous Settings.

1.1.2.1.1 Rule Execution Properties

The **Rule Execution Properties** entry in the **System Settings** dropdown list allows you to temporarily disable all of the rules in TeamConnect.

It is useful to disable rules during some processes. For example, when running an upgrade tool, rules that activate automatically during the upgrade may cause the upgrade tool to fail.



Rule Execution Properties

Both options are selected by default.

The options in the **Rule Execution Properties** section are described as follows:

Rule Execution Properties

Field	Description
Enable Rule Execution	Clear this check-box if you want to temporarily disable all of the rules in TeamConnect.
Enable Legacy Rule Execution	Clear this check-box if you want to temporarily disable all of the legacy rules in TeamConnect. The **Top Level/System/Legacy Rules** folder is used for storing Java class files for legacy rules that were written for TeamConnect. This centralized folder allows the files to be shared by multiple objects instead of being uploaded to each individual object definition.

Once you select the appropriate options, you must click **Save** in order for the settings to take effect.

Important: *If you temporarily disable rules and you want to re-enable them, you must go back to the **Rule Execution Properties**, select the appropriate check-box, and then click **Save**. If you want to enable legacy rule execution, you must select the **Enable Legacy Rule Execution** check-box and the **Enable Rule Execution** check-box. Legacy rule execution does not succeed unless both check-boxes are selected.*

For more information about rules, see Rule Components.

1.1.2.1.2 Miscellaneous Settings

If there is another system which your organization wants to use in tandem with TeamConnect, such as Microsoft Exchange or Lotus Notes, Mitratech can service a custom integration project for your organization.

In order for a TeamConnect to integrate with another system, the user must enter specific information into the available fields in the **Other Settings** page accessed from the user's **Preferences** page. However, these fields are unavailable to the user unless you create them.

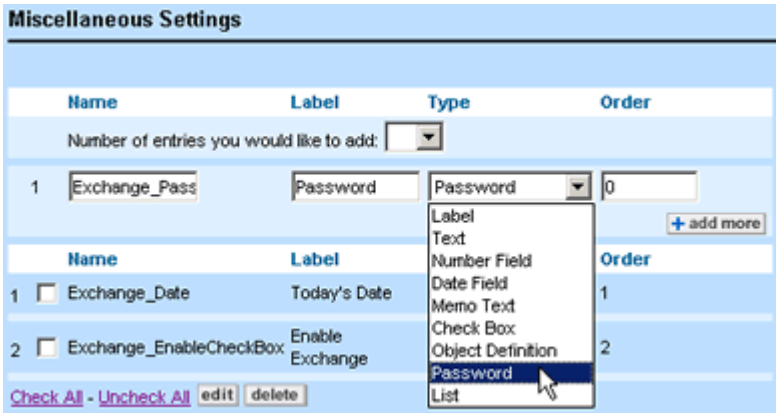
You may create custom fields for system integration in the **Miscellaneous Settings** screen. Depending on the system your organization wants to integrate, Mitratech may provide you with specific guidelines and information for creating custom fields for your integration. However, the procedure itself is the same as creating custom fields for an object definition.

To access the Miscellaneous Settings screen

Select **Misc. Settings** from the **Tools** drop-down list.

The corresponding Miscellaneous Settings screen appears.

The Misc. Settings tool is only available in the **Tools** menu if you have been given the right to it on the **Rights** tab in either your user account or a group account.



Miscellaneous Settings Tool Screen

The following table describes the items in the **Miscellaneous Settings** screen.

Miscellaneous Settings Tool Screen

Field	Description
Name	Enter an alphanumeric name (maximum 250 characters) to uniquely identify the field.
Label	Enter a label (maximum 250 characters) for the field as you would like it to display in the **Other Settings** page of the user's **Preferences** page.
Type	Select an option to specify the type of custom field that displays in the **Other Settings** page of the user's **Preferences** page. The options in this drop-down list are the same as the field types used for creating custom fields for objects, except for passwords. When creating a **Password** custom field for system integration, the following two fields appear on the user's **Other Settings** page of the **Preferences** page: • **New Password**—The user must enter the password required for the system that is integrating with TeamConnect. • **Confirm Password**—The user must confirm the password used for the system that is integrating with TeamConnect.
Order	Enter an integer to indicate the display **Order** of the custom field in the **Other Settings** page of the user's **Preferences** page. If you create more than one custom field for the user, the custom fields appear in the order specified, lowest number first. ***Note:*** *Items with the same* ***Order*** *are sorted and displayed alphabetically.*
add more	Click to add the custom field to the **Other Settings** page of the user's **Preferences** page.

To create a custom field for system integration

1. In the **Tools** drop-down list, select **Misc. Settings**.
2. Fill in the appropriate fields.

   If necessary, you can learn more about custom fields, their name requirements and types under Creating Custom Pages.
3. Click the **add more**.

   The custom field added is immediately available in the **Misc. Settings** tab of the user's **Preferences** screen.
4. Give users the Read and Update rights for **Preferences** in their user or group accounts, so that they can use the newly added fields.

Tip: *You can view and access the information entered by the user in their **Misc. Settings** tab by clicking the **Preferences** hyperlink in the user's account.*

1.1.2.2 Setting up Invoices and Line Items

Options and default behavior across all invoices records and for invoice record - Line Items sections can be configured from the Object Definitions area, using the following operations:

- Defining Non-US Tax Categories
- Setting up Invoice Duplicate Checking
- Setting up Invoice Line Items

1.1.2.2.1 Defining Non-US Tax Categories

If you use CSM, TeamConnect lets you define Non-US Tax categories to support e-Billing invoices that contain Non-US taxes (such as VAT or GST).

To define non-US tax categories

1. In the Designer, click **Go to**, and then click **Object Definitions**.
2. Click the **Invoice** object definition.
3. Click the **Non-US Tax Categories** tab.
4. In the **Show items node** drop-down list, select **Fee** or **Expense**.
5. In the **Item Name** field, enter the a unique name for this item.
6. In the **Order** field, enter a number to indicate the order in which this item appears to the user. Items with the same order number are sorted alphabetically.
7. In the **Tree Position** field, enter a four-character alphanumeric code that is unique to any other code.

8. In the **Tax Code** field, enter the name for the tax code that applies to this item.

   Item Name, **Tree Position**, and **Tax Code** only need to be unique for items of the same type.

9. Click **+add more**.
10. To add more items, click +add more and repeat steps 4 through 8.
11. Click **Save** or **Save and Close**.

You will be able to create financial accounts in the TeamConnect user interface to track Non-US Taxes separately. The Non-US Tax category **Item Name** values appear in the **Post Non-US Tax of Type** drop-down list on the Accounts **Posting Criteria** page.

1.1.2.2.2 Setting up Invoice Duplicate Checking

A system setting will determine whether invoices will be checked for duplicates. For more information about invoice duplicates and versions, see Invoice Duplicates and Invoice Versions.

To set up invoice duplicate checking

1. From the **System Settings** drop-down list, select **Backward Compatibility Settings**.
2. Do one of the following actions:
3. To allow duplicate invoices, select the **Allow Duplicate Invoices (Same invoice number, invoice date year and vendor)** check-box.
4. To check for and prevent duplicate invoices, clear the **Allow Duplicate Invoices (Same invoice number, invoice date year and vendor)** check-box.
5. Click **Save**.

1.1.2.2.3 Setting up Invoice Line Items

This section describes how to set up the ways users will work with Invoice Line Items.

Note: *The terms "fee" and "task" are used interchangeably for Invoice Line Items. For example, in the user interface, Line Item types include Fee. In the developer's view for Line Item object definitions, Task Categories refer to categories for Fee Line Items.*

1.1.2.2.3.1 Setting up General Invoice Line Items Behavior

General Invoice Line Items behavior includes:

- Setting up which Line Item fields display in Concise view
- Setting up which existing Line Item field values can be copied to a new Line Item
- Setting up whether Line Item discount adjustments are applied by flat-rate or as a percentage

To set up general Line Item fields behavior

1. In the **Go to** drop-down list, select **Object Definitions**.
2. Expand **Invoices** and click the **Line Items** link.
3. Select the **Line Items** tab.
4. To set up which fields display on an invoice **General** page, **Line Items** section area for line item entry (in edit-mode), do one of the following:
   - For each field to display in the invoice **Line Items** section (for line item entry), from the corresponding **Line Item Entry** column drop-down list, select **Show**.
   - For each field to hide in the invoice **Line Items** section (for line item entry), from the corresponding **Line Item Entry** column drop-down list, select **Hide**.
5. To set up which fields display on an invoice **General** page, **Line Items** section area for existing line items (in edit-mode), do one of the following:
   - For each field to display in the invoice **Line Items** section (table for existing line items), from the corresponding **Table Display** column drop-down list, select **Show**.
   - For each field to hide in the invoice **Line Items** section (table for existing line items), from the corresponding **Table Display** column drop-down list, select **Hide**.
6. For each **Line Item** field whose existing value to copy to a blank Line Item block using the **Copy values from previous line** link, check the corresponding **Copy Capability** check-box.
7. Click **Save**.

Setting the Discount Application Type for Line Item Adjustments

When users adjust Invoice Line Items, an Original Discount amount can be applied in one of the following ways:

- As a flat amount
- As a percentage

By default, Discounts are applied to Line Item adjustments as flat amounts. The Original Discount value will be automatically copied to the Current Discount field (Batch Adjustment mode) or will be displayed in the Current Discount field of the **Invoice - Adjust Line Items** screen (Detail Adjustment mode) after invoice changes are saved.

When Discounts are applied as percentages, the percentage is calculated in the following way:

Discount % = (Original Discount) * 100 / [(Original Rate) * (Original Hrs/Units)]

Based on Line Item Object Definition settings, a Line Item can be adjusted using Current Rate, Current Hrs/Units, Current Disc (Batch Adjustment only), Current Amount.

The Current Discount will be applied as the percentage value only if Current Rate or Current Hrs/Units fields are adjusted.

The Current Discount is not applied as a percentage, if a Line Item is adjusted by any of the following methods:

- The Current Amount is manually adjusted.
- The Current Discount is manually adjusted.
- The Current Amount is indirectly adjusted by an Invoice Header adjustment.

For more information, see Adjusting Line Items with Discounts.

To toggle between setting adjustment discounts as a flat amount or percentage

1. In the **Go to** drop-down list, select **Object Definitions**.
2. Click the **Invoices** link.
3. From the **General** tab, do one of the following actions:
    - To apply the adjustment discount as a percentage, check **Discounts are applied as percentage of the charge amount**.
    - To apply the adjustment discount as a flat rate value, uncheck **Discounts are applied as percentage of the charge amount**.
4. Click **Save**.

1.1.2.2.3.2 Setting up Invoice Line Items Adjustment Behavior

Invoice Line Items Adjustment Behavior includes:

- Setting up whether Line Items can be adjusted in Batch Adjustment mode or in Detail Adjustment mode
- Setting up which Line Item fields will be read-only in Batch Adjustment mode
- Entering adjustment reasons

Invoice Line Items Adjustment Options

You may adjust invoice Line Items as follows:

- Detail Adjustment: (recommended for Invoices with few Line Items) For TeamConnect 2.4 SP3 and earlier, Line Items have been adjusted in this mode. When enabled adjusters must select each Line Item, adjust the Line Item, and save changes to individual Line Items separately.
- Batch Adjustment: (recommended for Invoices with many Line Items) When enabled adjusters can make all Line Items for an invoice adjustable with one button click, adjust the Line Items, and save changes to multiple Line Items with one button click. This option is helpful for tabbing through multiple Line Items.
- In-Line & Bulk Adjustment: (recommended for Invoices with many Line Items) Introduced in TeamConnect 3.4 SP1, this setting allows adjusters to make changes to multiple Line Items at a time, with improved screen navigation, the ability to add comments to Line Items directly from the In-Line & Bulk Adjustment (IBA) screen, and other additional features.

To select between Individual Invoice Line Items Adjustment, Batch Adjustment, or In-Line & Bulk Adjustment

1. In the **Go to** drop-down list, select **Object Definitions**.
2. Expand the **Invoices** link and click **Line Item**.
3. Do one of the following actions:
   - To adjust invoice record line items individually, from the **Default Adjustment Mechanism** drop-down list, select **Detail Adjustment** (default).
   - To adjust invoice record line items in batch mode, from the **Default Adjustment Mechanism** drop-down list, select **Batch Adjustment**. This option allows you to adjust multiple Line Items simultaneously and save changes with one button click.
   - To adjust invoice record line items in in-line & bulk mode, from the **Default Adjustment Mechanism** drop-down list, select **In-Line/Bulk Adjustment**. This option allows you to adjust multiple Line Items simultaneously and add comments to individual Line Items from the same screen.
4. Click **Save**.

Setting Batch Adjustment Line Item Fields Read/Write Permissions

If Line Items are set to Batch Adjustment mode, you can restrict which Invoice Line Item fields can be used to adjust invoices by setting certain fields to read-only status.

Note: *The following description only applies if you have set the Line Items **Default Adjustment Mechanism** to **Batch Adjustment**.*

Fields that can be set to read-only include:

- Current Total
- Current Discount
- Current Qty
- Current Rate

To set Batch Adjustment Line Item fields to read-only

1. In the **Go to** drop-down list, select **Object Definitions**.
2. Expand the **Invoices** link and click **Line Item**.
3. Select the **Line Items** tab.
4. For each **Line Item** field (**Current Total**, **Current Discount**, **Current Qty**, **Current Rate**) to set as read-only when performing a batch adjustment, check the corresponding **Read Only in Adjustment** check-box.

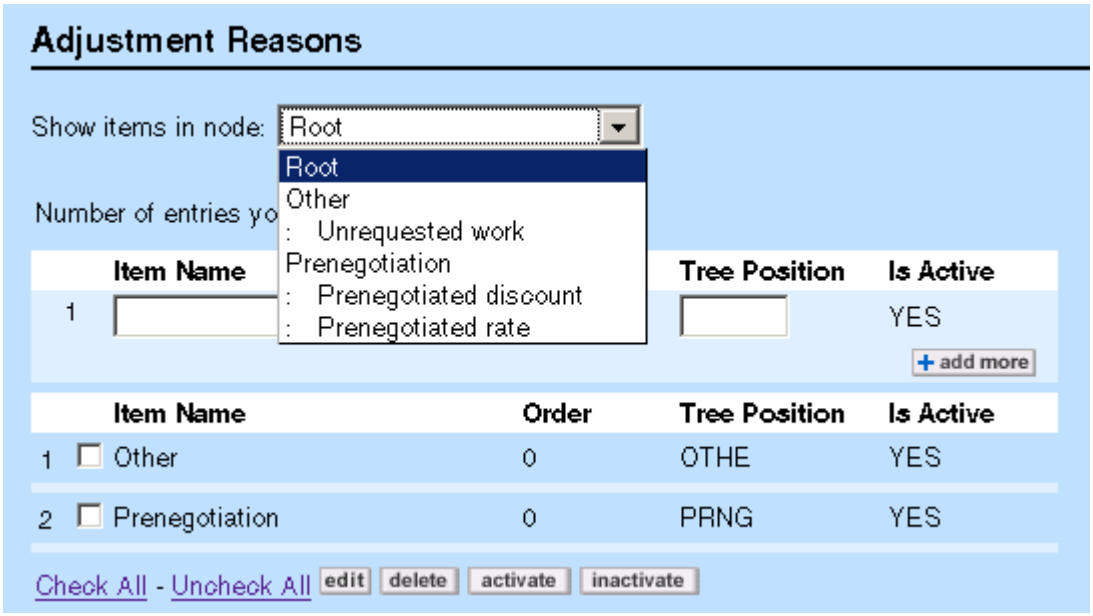
At least one of the adjustable Line Item fields above needs to be left as non-read only. Users need to be able to adjust at least one field.

5. Click Save.

To access the Adjustment Reasons page

1. In the **Go to** drop-down list, select **Object Definitions**.
2. Expand the **Invoices** link and click **Line Item**.
3. Select the **Adjustment Reasons** tab.

Adjustment reasons are organized in a parent-child hierarchy. Initially there is only one node, "Root", and any adjustment reasons that you enter become children of the "Root" node. If you want to enter reasons that are children of some other node, choose that node name from the **Show items in node** drop-down list, as shown below.



Adjustment Reasons Hierarchy

To work with Adjustment Reasons

1. Choose a node from the **Show items in node** drop-down list.
2. The adjustment reasons that are children of that node are shown.
3. If you wish to delete an existing reason, click its checkbox and click **delete**.
4. If you wish to inactivate an existing reason, click its checkbox and click **inactivate**. Inactive reasons won't appear to the end user who is doing an adjustment.
5. If you wish to reactivate an existing reason, click its checkbox and click **activate**.

6. To edit an existing reason, click its checkbox and click **edit**.

   The **Name** and **Order** fields become editable and you can change their values. When finished, click **OK**.

7. To enter a new adjustment reason, begin typing in the editable fields above the existing items.

   - Enter a **Name** for the adjustment reason.
   - Enter an **Order** value. **Order** determines the sequence in which the reasons appear to the end user.
   - Enter a **Tree Key**. This value must be unique, different from all other adjustment reasons.
   - **Is Active** is presumed to be Yes when a reason is first entered. You can subsequently **inactivate** it after it is saved.
   - When finished, choose one of the **Save** buttons at the top of the page. Clicking **Add More** will also save your work and present you with new editable fields for additional adjustment reasons.

1.1.2.3 Using Duplicate Contact Manager

Duplicate Contact Manager enables TeamConnect designers, developers, and other highly trained IT specialists to identify duplicate contact records. A Contact Sweeper Tool is provided allowing the tool users to select which duplicate contact records to keep and which contact records to remove from the system.

Contact Sweeper also allows for the redirection of existing pointers to duplicate contact records, to reference the contacts you choose to save. Duplicate Contact Manager may also be used to prevent end users from creating or updating contact records that are intentional duplicates.

Duplicate Contact Manager includes:

- A Contact Sweeper Tool that can identify potential duplicate contact records based on specified contact fields and sample contact data values.
- A duplicate contact checking option with a settings screen to select which contact fields should contain unique data across records. After duplicate contact checking is enabled, users will be warned when they create/update contacts with possible existing duplicate records.

Duplicate Contact Manager must be configured by a solution developer, but the Contact Sweeper can be invoked by any user who is granted the right to that tool.

It is recommended that you use Duplicate Contact Manager in the following order:

1. Using Contact Sweeper
2. Preventing Duplicate Contacts

1.1.2.3.1 Using Contact Sweeper

Contact Sweeper helps identify, replace, and remove contacts that are duplicates of other existing contacts. Since the contact attribute is found in a large number of database tables, the process of manually changing a contact throughout TeamConnect is complex. Contact Sweeper hides that complexity and makes the process faster and more reliable.

1.1.2.3.1.1 Prerequisites

Contact Sweeper is a system tool and, like other system tools, it requires the user running it to have sufficient authority. You must use caution when granting rights to this tool because its use may greatly alter the information in the database.

To give a user group rights to the Contact Sweeper tool

1. Click **Groups** in the **Admin** tab of TeamConnect.
2. Click on the specific group to give rights to the Contact Sweeper tool.
3. Click the **Tool Rights** tab of the group's settings.
4. Click the **Edit** button.
5. Select the check-box for **Contact Sweeper**.
6. Save the group's tool rights settings.

Note: *If the Contact Sweeper window is prematurely closed before steps have finished running, all steps done in the session must be repeated.*

It is essential that you back up your database before running Contact Sweeper. Contact Sweeper makes extensive changes to the TeamConnect database. After you have run the Fix References step in Contact Sweeper, the only way to reverse its effects is by restoring the database from the backup.

Important: *Several prompts in Contact Sweeper may display in a pop-up window. If your web browser is configured to not allow pop-up windows, you must reconfigure it to allow pop-ups from the TeamConnect web server.*

Using Contact Sweeper entails the following steps:

- **Step 1 of 5 (Generate Report)—**Specifies the conditions to be used in searching for duplicates, then performs the search and prepares an Excel spreadsheet containing the generated report, which you then save.
- **Step 2 of 5 (Edit Report)—**Indicates which duplicate contacts must be replaced by other contacts. Do this by editing the values in the generated report spreadsheet (offline step).
- **Step 3 of 5 (Upload Report)—**Uploads your edits to TeamConnect and makes them available for the following steps.
- **Step 4 of 5 (Fix References)—**Performs an optional preview and performs the actual replacement of references.

- **Step 5 of 5 (Delete Duplicates)—**Deletes the contacts whose references were replaced in the previous step.

You may do all of these steps, except step 2, using the Contact Sweeper tool window.

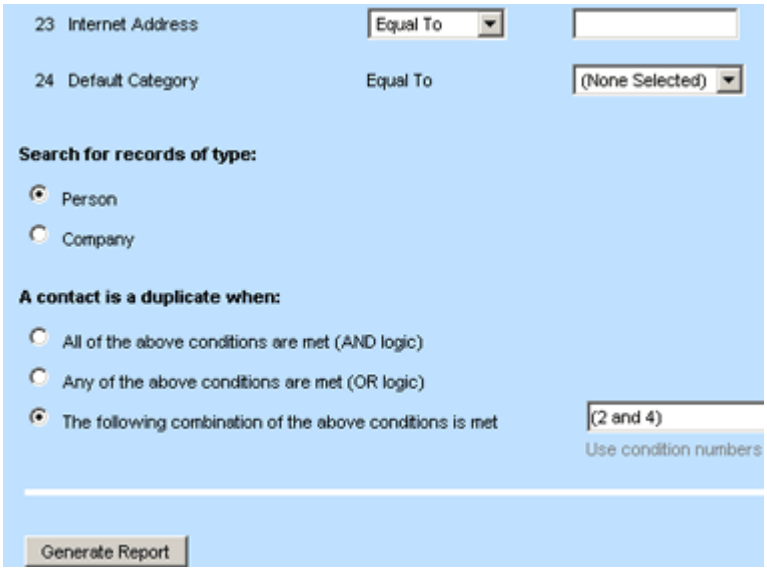
To start the Contact Sweeper tool

Select **Contact Sweeper** from the **Tools** drop-down list in Designer.

1.1.2.3.1.2 Generate Report (Step 1 of 5)

This step creates a report of all contacts that meet the conditions you specify. You may run this step repeatedly, specifying a different set of conditions each time. Contacts meeting your conditions appear in an Excel spreadsheet.

Generate Report Interface (1 of 2)



Generate Report Interface (2 of 2)

Contact Sweeper displays some of the possible criteria. All the possible criteria are available for your use in specifying conditions. Ordinarily, only some, not all, of the criteria are actually used. For example, your conditions may test for equality with a criterion, a partial string match, and other conditional expressions.

Option buttons specify whether to form a match using all conditions, any of the specified conditions, or some custom combination of conditions. In the previous example, a custom combination is used, specifying that condition 2 (First Name match) and condition 4 (Last name / Company Name match) must both be true.

The initial conditions that appear when you run this step are those that were used the last time you ran this step. However, you have the ability to change any condition that you wish.

Points To Remember

Keep the following points in mind when setting up Contact Sweeper reports:

- Criteria that are left blank will be ignored when generating the report.
- Try to keep your conditions restrictive, so a relatively small set of contacts is returned. You may wish to search on a last name plus postal code, or other such conditions that should yield a small number of results.

  Note: *You may be even more restrictive by searching on unique identifiers such as Social Security number or driver's license number. If your conditions are defined too loosely, many contacts will be returned, and it can be difficult to find the true duplicates.*

- Some of the criteria may relate to attributes that have multiple values per contact. For example, a contact can have multiple addresses, multiple phone numbers, and multiple email addresses. When you use a criterion like these to specify a condition, be aware that only the default value for the contact is tested. Contact Sweeper does not search through all of the multiple values for the contact, just the default.

- Contact Sweeper understands the difference between a contact that represents a company and a contact that represents a person. One of the search conditions is whether you are looking for persons or companies.

  For example, if your search involves the person name "Pierre Cardin", the company "Pierre Cardin" does not appear in the report on potential duplicates. Similarly, if you have asked to search for companies, but you have also specified conditions that apply only to persons (such as a First Name value), those conditions are ignored.

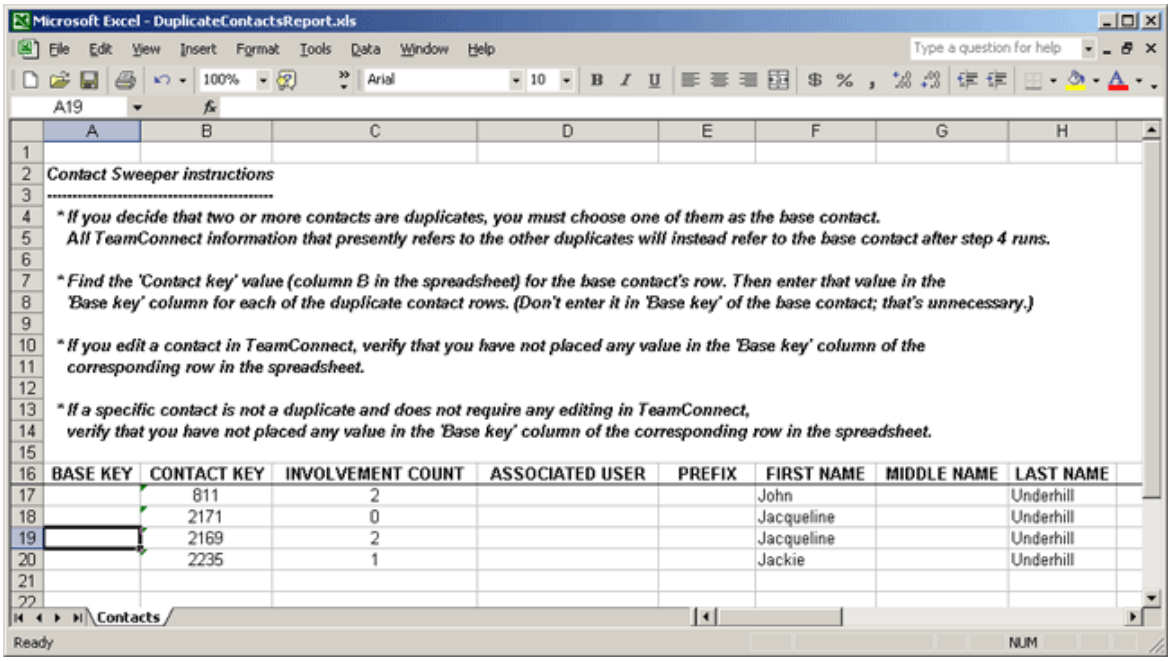
Generating a Report

Generating a report operates as follows:

- After you click **Generate Report** and Contact Sweeper performs its search, you are prompted to save the resulting "duplicate reference report" as an Excel spreadsheet. The default name of the report is **DuplicateContactsReport.xls**.
- Choose an easily remembered name and location for this file. It is possible to store many different sets of results by running this step repeatedly, using different conditions each time, and saving each resulting spreadsheet with a different name.
- In some web browsers, the spreadsheet may open automatically. If so, you must save it, using a name and location that you remember. Other browsers may download the spreadsheet automatically without prompting you. If nothing appears to happen after clicking the **Generate Report** button, check your web browser's default download location for **DuplicateContactsReport.xls**.

1.1.2.3.1.3 Edit the Report (Step 2 of 5)

The spreadsheet produced by the Generate Report step contains one row per contact. Contact Sweeper has identified all the contacts in the spreadsheet as being potential duplicates of each other, because they all match the conditions you specified in step 1. Review these contacts and decide which ones are actual duplicates.



Report Editor Interface

Review and edit the report as follows:

- If the spreadsheet from step 1 is not already open, open it now.

  The spreadsheet contains blank cells in column A ("Base key"), the primary key of the contact in column B ("Contact key"), then a large number of contact information fields in the following columns. The columns "Involvement count" (indicating how many projects or other objects refer back to this contact) and "Associated user" (indicating that this contact is related to a TeamConnect user) are useful in deciding which contacts to keep and which to replace.

  If you decide that two or more contacts are duplicates, you must choose one of them as the base contact. All TeamConnect information that presently refers to the other duplicates will instead refer to the base contact after step 4 runs.

- Find the "Contact key" value (column B in the spreadsheet) for the base contact's row. Then enter that value in the "Base key" column for each of the duplicate contact rows. (Don't enter it in "Base key" of the base contact; that's unnecessary.)

  Note: *Most of the cells in the spreadsheet are protected. Do not turn off this protection, or you may cause the spreadsheet to become unusable. Only the "Base key" column of the spreadsheet is editable. You cannot move, insert, or delete rows and columns.*

  In the example shown above, the row with contact key 2169 is the base contact that will remain after Contact Sweeper completes. Therefore, you would type 2169 in the Base Key cell for contacts 2171 and 2235. Contact 811 is a completely different person, not a duplicate, so we make no entry in that row.

  Note: *Contacts where the column "Associated user" contains a value will not be replaced or deleted, even if you request this.*

- Instead of replacing some of the potential duplicate contacts, you may want to edit them so that they do not appear as duplicates anymore. If you edit a contact in TeamConnect, verify that you

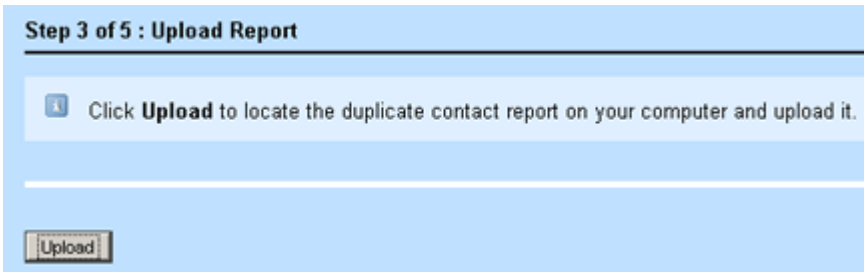
have not placed any value in the "Base key" column of the corresponding row in the spreadsheet.

- If a specific contact is not a duplicate and does not require any editing in TeamConnect, verify that you have not placed any value in the "Base key" column of the corresponding row in the spreadsheet.
- Save your changes to the spreadsheet.

1.1.2.3.1.4 Upload Report (Step 3 of 5)

When you have finished all your data entry from the previous step, upload the spreadsheet to TeamConnect as follows:

1. Click the **Upload** button.
2. In the resulting **Upload New File** window, type the full path and file name in the **File Name** field or use the **Browse** button to locate the file on your computer.
3. Click upload file to move your spreadsheet into TeamConnect.



Upload Report Interface (1 of 2)

Upload Report Interface (2 of 2)

Now you are ready to run the Fix References step.

1.1.2.3.1.5 Fix References (Step 4 of 5)

The Fix References step provides you with the option to preview the database changes that are about to occur, then updates the TeamConnect database with the contact changes you requested in the spreadsheet.

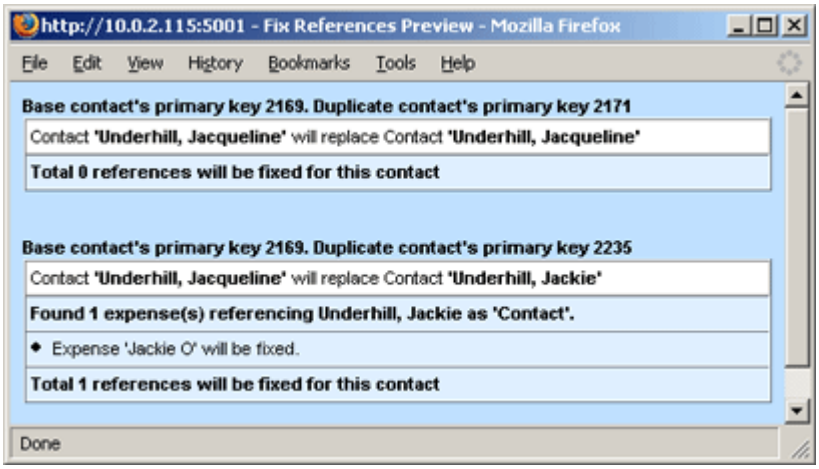
Caution: *Perform this step when there are no other active TeamConnect users working with your TeamConnect database. This step puts the database into an inconsistent state until it*

completes. It is strongly recommended that you back up your database before performing this step. The changes that are done in this step are not reversible, except by restoring a database backup.

Important: *Custom rule processing could potentially interfere with the database updates done by this step. Deactivate all rules that relate to the Contact object before performing this step. When the step is complete, you may activate the rules again.*

Preview the suggested changes and update the database as follows:

- Click **Preview** to display a pop-up window with text that describes the proposed database changes. No actual changes are done during Preview.



Fix References Preview Screen

In the previous example, there were references in existence for each duplicate contact. However, it is possible to have duplicate contacts that have never been used as a reference in any other record.

- Click **Fix References** to apply the changes in the spreadsheet to the database. Read the confirmation pop-up window, determine if you wish to continue, and click **OK** to continue or **Cancel** to stop.

  This step may take some time, depending on the number of duplicate contacts and references that must be replaced. When finished, the Contact Sweeper screen refreshes and a completion message will appear below the **Fix References** button.

Contact Sweeper Completion Message

Contacts can become references in several other types of TeamConnect objects. Contact Sweeper fixes the following kinds of references:

- Every contact-related role in every contact-centric custom object record.
- Contacts that appear in the Relations tab of other Contact records.

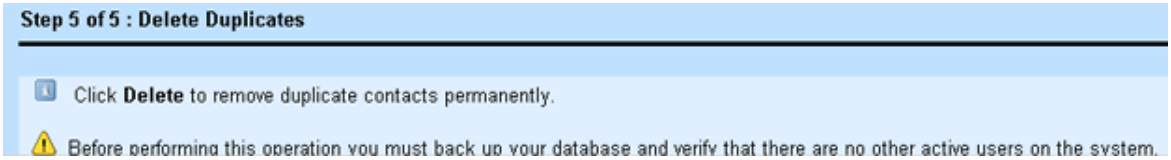
- Contacts that appear as "Vendor" or "Involved" in the **Posting Criteria** tab of an Account record.
- The "Contact" reference in an Expense record.
- The "Contact" reference in a Task record.
- The "Vendor" reference in an Invoice record.
- The "Vendor" and "Timekeeper" references in an Invoice Line Item record.
- The "Involved" references in a Project record.

By default, the changes made during the Fix References process are written to the system log at the Debug level. This level provides complete information about the changes that have been made.

It is possible that you have previously set the system logging level to some more restrictive level, such as Warn, using the **System Preferences** screens. If so, logging occurs at that more restrictive level, and some of the detailed information is lost. When you determine the level of logging you want during Contact Sweeper operation, verify that the system logging level setting is compatible with your needs.

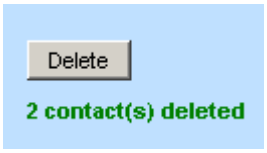
1.1.2.3.1.6 Delete Duplicates (Step 5 of 5)

Click **Delete** to delete the contacts that have just been replaced by the base contact. Answer **Yes** to the confirmation pop-up message.



Delete Duplicates Interface

After the deletion operation, the Contact Sweeper screen refreshes, and a success message indicating the number of contacts deleted appears below the **Delete** button.



Confirmation Message

Note: *The count reflects the actual number of contacts deleted. This number may be less than the number of contacts listed in step 4, if one or more of those contacts is associated with a user account. Such contacts are not deleted by Contact Sweeper. The system log notes such cases.*

Note: *If you click* ***Delete*** *without first doing* ***Fix references****, TeamConnect attempts to delete all the contacts for which you have entered a value in the "Base key" column of the spreadsheet. But since these contacts were not fixed yet, many of them may have references, such as being used in an "involved" role. In such cases, any contact with references is not deleted, and an error message is entered in the log file noting each case.*

1.1.2.3.2 Preventing Duplicate Contacts

The Duplicate Contact Manager can perform day-to-day duplicate contact checking and prevent the creation or update of duplicate contact records. You must select which contact fields require unique field data across contact records and activate the duplicate contact checking setting.

Afterwards, when a user creates or updates a contact, TeamConnect compares the new contact's data against existing contacts to determine whether duplicates exist and warn the user.

If you activate the Duplicate Contact check for day-to-day usage, the following activities are affected:

- Creating a new contact from the **Create a new Contact** screen
- Updating an existing contact (person or company)
- Creating a new contact (person or company) from a Contact search module pop-up window
- Creating a new contact from a wizard using a contact search module (**Search for Contacts** pop-up window)
- Creating a new contact through the XML layer

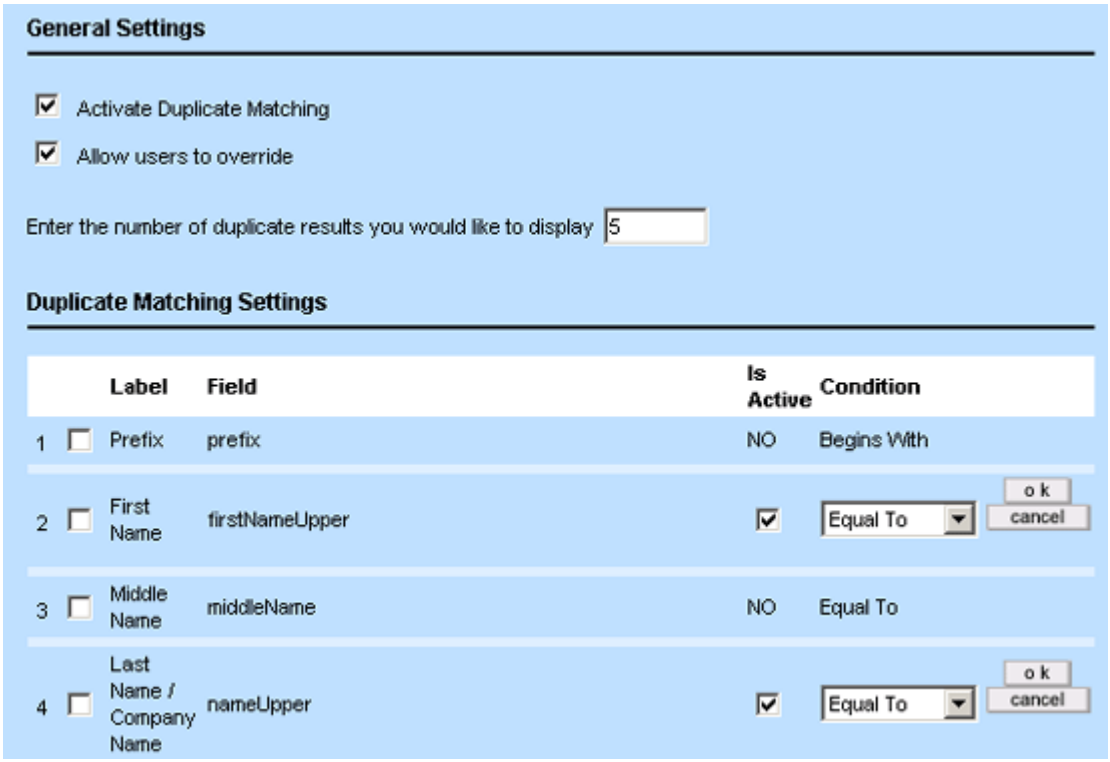
1.1.2.3.2.1 Activating Duplicate Contact Checking

Consider the following before using the Duplicate Contact Manager and Contact Sweeper:

- Contacts that are updated or added through the actions of wizards, rules, and custom views are not automatically checked for duplicates. If you wish to prevent potential duplicates that result from these actions, you must modify your wizards, rules, and views. For wizards and custom views, one option is to add the **Potential Duplicates** block to your wizard or view. Also, the `ContactSerivce.readDuplicateContacts` API method is available for use in your custom code. This method allows you to check for duplicates before committing a contact record. See Using Rules for more information.
- Review the design of wizards, rules, and custom objects to verify that the criteria for identifying duplicate contacts is narrow enough not to prevent users from completing wizards successfully. You may need to modify existing rules and wizards to handle duplicate exceptions wherever a contact is created or updated.
- After you use the Contact Sweeper to remove existing duplicate contact records from TeamConnect, you should determine which contact fields should be required to create a contact and for day-to-day duplicate contact checking. For example, a person contact might require the **First Name**, **Last Name**, and **Phone Number** fields. This standard could be enforced by creating validation rules. Company contacts might require the **Company Name**, **Phone Number**, **Street Address**, and **Postal Code** fields.

The contact fields you set as Active from the Duplicate Contact Manager settings screen determine which contact fields require unique data when users create or update contact records (when duplicate contact checking is enabled).

Note: *You may select one or more contact fields to use for search criteria.*



Duplicate Contact Manager Settings Screen

The Duplicate Contact Manager settings screen determine which contact fields are used to find potential duplicate contact records.

To select a contact field to use for duplicate contact checking

1. From the Designer **Go To** drop-down list, select **Object Definitions**.
2. Select the **Contact** object definition.
3. Select the object definition's **Duplicate Contact Manager** tab.
4. Select the far left check-box for the contact field to use for duplicate contact checking.
5. Click **edit**.
6. Select the **Is Active** check-box.
7. Select the value-matching condition (for example, **Equal To**).
8. Click **ok**.

In the following steps, you can:

- Select whether to enable day-to-day duplicate contact checking (and whether to enable a user override to allow creation of duplicate contact records)
- Select which contact fields to activate for comparison of a new contact's field data against existing contacts' data
- Select logic for combining multiple contact fields as duplicate record identifiers

To activate duplicate contact checking

1. From the Designer **Go To** drop-down list, select **Object Definitions**.
2. Select the **Contact** object definition.
3. Select the **Object Views** tab.
4. Choose an object view. On the General tab of the object view, ensure that the block **Duplicate Contacts** is used by the object view. If it is not, add that block to the object view.
5. Optionally, repeat the previous step for as many object views as are affected by duplicate contact checking.
6. Select the object definition's **Duplicate Contact Manager** tab.
7. To enable day-to-day duplicate contact checking, select the **Activate Duplicate Matching** check-box. If users try to create or update a contact that match existing duplicate contact records, a warning message will display.

   To allow users to override duplicate contact checking and continue with the create or update operation, select the **Allow users to override** check-box.
8. To set the maximum number of potential duplicate contacts results that display during day-to-day duplicate contact checking, type the number in **Enter the number of duplicate results you would like to display**. The default value is 20.

   If during a contact creation operation, the Duplicate Contact Manager finds more potential duplicate contacts than the maximum duplicate results value you set above, a warning message will display. For example, **25 possible duplicate(s) found. Displaying 20 of 25**.
9. From the **Duplicate Contact Manager** tab, check one or more of the following Labels and click **edit**. Select the **Is Active** check-box, and select a **Condition** from the drop-down menu to use the corresponding contact field for duplicate matching. Click **ok** to save your selections for the corresponding contact field.

   A full list of available items is described in the Duplicate Matching Settings table. Labels and Conditions marked with an asterisk are recommended for duplicate matching.
10. If you selected multiple contact fields above, select one of the following for **A contact is a duplicate when**:
    - **All of the above conditions are met (AND logic)**—Detects duplicate contacts if the values of all selected fields above match between the contact records being compared
    - **Any of the above conditions are met (OR logic)**—Detects duplicate contacts if the values of any selected fields above match between the contact records being compared
    - **The following combination of the above conditions is met**—Type a custom logical rule for contact field. For example, to compare contacts by (First Name, Last Name, and Social Security Number) or (Default Postal Code and Phone number), type **(2 and 4 and 12) or (17 and 20)**

      The field numbers you enter when using **The following combination of the above conditions is met** must refer to contact fields that are set as **Active**.
11. Click **Save**.

Duplicate Matching Settings

Label	Field	Condition
Prefix	prefix	Begins With *Equal To Contains Ends With Has No Value Has Value
***First Name**	firstNameUpper	Begins With *Equal To Contains Ends With Has No Value Has Value
Middle Name	middleName	Begins With *Equal To Contains Ends With Has No Value Has Value
***Last Name / Company Name**	nameUpper	Begins With *Equal To Contains Ends With Has No Value Has Value
Suffix	suffix	Begins With *Equal To Contains

		Ends With Has No Value Has Value
Nickname/Alias	aliasUpper	Begins With *Equal To Contains Ends With Has No Value Has Value
Company	company	Equal To
Title	title	Begins With *Equal To Contains Ends With Has No Value Has Value
ID	numberStringUpper	Begins With *Equal To Contains Ends With Has No Value Has Value
***Driver's License**	driverLicense	Begins With *Equal To Contains Ends With Has No Value Has Value

***SSN / Tax ID**	SsOrTaxNumberString	Begins With *Equal To Contains Ends With Has No Value Has Value
***Birth Date**	birthDate	*Equal To More Than X Days After During X Days After During X Days Before More Than X Days Before Between X and Y Days
Street	defaultAddress_Street	Begins With *Equal To Contains Ends With Has No Value Has Value
City	defaultAddress_City	Begins With *Equal To Contains Ends With Has No Value Has Value
State	defaultAddress_State	Begins With *Equal To Contains Ends With

		Has No Value Has Value
Postal Code	defaultAddress_PostalCode	Begins With *Equal To Contains Ends With Has No Value Has Value
County	defaultAddress_County	Begins With *Equal To Contains Ends With Has No Value Has Value
Country	defaultAddress_CountryItem	Equal To
Phone	defaultPhone_PhoneString	Begins With *Equal To Contains Ends With Has No Value Has Value
Fax	defaultFax_FaxString	Begins With *Equal To Contains Ends With Has No Value Has Value
Email	defaultEmail_EmailString	Begins With

		*Equal To Contains Ends With Has No Value Has Value
Internet Address	defaultInetAddress_InetAddressString	Begins With *Equal To Contains Ends With Has No Value Has Value
Default Category	defaultCategory	Equal To

Using AND/OR Logic

If AND logic or OR logic is selected, when a user is creating/updating contacts, the duplicate check only evaluates field criteria that apply to the type of contact being created/updated.

For example, suppose duplicate contact checking is enabled for First Name, Last Name/Company Name, and Social Security/Tax ID using AND logic. If a user creates a company contact, duplicate contact checking performs a comparison against existing company records using the Company Name and Tax ID fields (ignoring the First Name criteria).

If custom logic is enabled, for example, "(First Name, Last Name/Company Name, and Social Security Number/Tax ID) or (Default Postal Code and Phone Number)," when creating/updating a company contact, only "(**Last Name**/**Company Name** and **Social Security Number**/**Tax ID**) or (Default Postal Code and Phone Number)" is checked.

Creating Contacts in the Search for Contacts Pop-up Window

When users create contacts from a **Search for Contacts** pop-up window, duplicate contact checking only compares certain fields to existing contact records (for example, when an Involved record is added to a project).

For example, when a user creates a contact from the **add person** button, the following fields' values are compared to existing contacts: **First Name**, **Last Name**/**Company Name**, **Title**, **ID**, **SSN**/**Tax ID**, default **Street Address**, default **City**, default **State**, default **Country**, default **Postal Code**, and default **Category** field.

For another example, when a user creates a company contact from the **add company** button, the following fields' values are compared to existing contacts: **Last Name**/**Company Name**, **SSN**/**Tax ID**, default **Street Address**, default **City**, default **State**, default **Country**, default **Postal Code**, and default **Category** field.

As an example of a person from the contact screen, you might set up the Duplicate Contact Manager settings screen as follows:

- **Activate Duplicate Matching** has been selected.
- **Allow users to override** has been selected.
- **Enter the number of duplicate results you would like to display** is set to 5.

2	☐	First Name	firstNameUpper	YES	Equal To
3	☐	Middle Name	middleName	NO	Equal To
4	☐	Last Name / Company Name	nameUpper	YES	Equal To
5	☐	Suffix	suffix	NO	Equal To
6	☐	Nickname/Alias	aliasUpper	NO	Equal To
7	☐	Company	company	NO	Equal To
8	☐	Title	title	NO	Equal To
9	☐	ID	numberStringUpper	NO	Equal To
10	☐	Driver's License	driverLicense	NO	Equal To
11	☐	SSN / Tax ID	SsOrTaxNumberString	NO	Equal To
12	☐	Birth Date	birthDate	NO	Equal To
13	☐	Street	defaultAddress_Street	NO	Equal To
14	☐	City	defaultAddress_City	NO	Equal To
15	☐	State	defaultAddress_State	NO	Equal To
16	☐	Postal Code	defaultAddress_PostalCode	YES	Equal To
17	☐	County	defaultAddress_County	NO	Equal To
18	☐	Country	defaultAddress_CountryItem	NO	Equal To
19	☐	Phone	defaultPhone_PhoneString	YES	Equal To
20	☐	Fax	defaultFax_FaxString	NO	Equal To
21	☐	Email	defaultEmail_EmailString	NO	Equal To
22	☐	Internet Address	defaultInetAddress_InetAddressString	NO	Equal To
23	☐	Default Category	defaultCategory	NO	Equal To

Check All - Uncheck All edit

A contact is a duplicate when:

○ All of the above conditions are met (AND logic)

○ Any of the above conditions are met (OR logic)

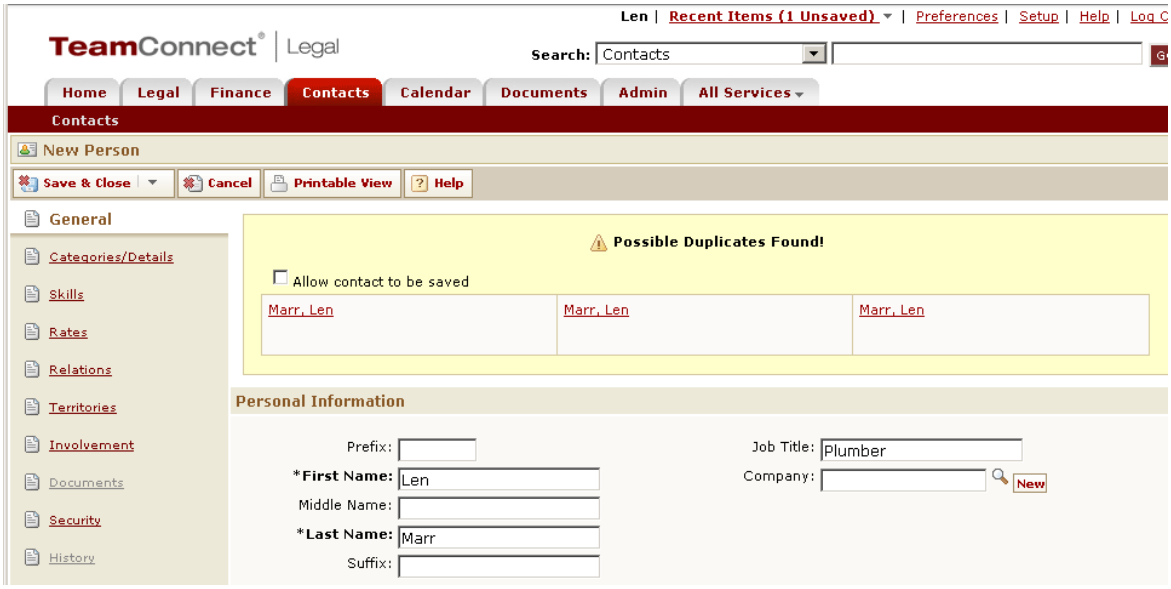
◉ The following combination of the above conditions is met (2 and 4 and 19) or (2 and 4 and 16)

Duplicate Matching Settings

In this example, a user may try to create a contact (type person) with the following field data:

- **First Name** - Len
- **Last Name** - Marr
- **Default Phone Number** - 213-380-5512

In this case, when the user clicks **Save**, a message appears showing possible duplicates that have been found:



Possible Duplicate Contacts Found Message Example 1

Note: *A message like "Displaying 5 of 7" (not shown in this example) would indicate that the Duplicate Contact Manager was configured to display a maximum of 5 duplicate results but 2 additional possible duplicates were found and not displayed.*

The user may perform one of the following actions:

- Override the duplicate contact check by selecting the **Allow contact to be saved** check-box and clicking **Save**.
- Click an existing contact link to verify that the contact is actually unique.

As an example of the contact screen, you might set the Duplicate Contact Manager settings screen as follows:

- **Activate Duplicate Matching** is selected.
- **Allow users to override** is selected.
- **Enter the number of duplicate results you would like to display** is set to 5.

2	☐	First Name	firstNameUpper	YES	Equal To
3	☐	Middle Name	middleName	NO	Equal To
4	☐	Last Name / Company Name	nameUpper	YES	Equal To
5	☐	Suffix	suffix	NO	Equal To
6	☐	Nickname/Alias	aliasUpper	NO	Equal To
7	☐	Company	company	NO	Equal To
8	☐	Title	title	NO	Equal To
9	☐	ID	numberStringUpper	NO	Equal To
10	☐	Driver's License	driverLicense	NO	Equal To
11	☐	SSN / Tax ID	SsOrTaxNumberString	NO	Equal To
12	☐	Birth Date	birthDate	NO	Equal To
13	☐	Street	defaultAddress_Street	NO	Equal To
14	☐	City	defaultAddress_City	NO	Equal To
15	☐	State	defaultAddress_State	NO	Equal To
16	☐	Postal Code	defaultAddress_PostalCode	YES	Equal To
17	☐	County	defaultAddress_County	NO	Equal To
18	☐	Country	defaultAddress_CountryItem	NO	Equal To
19	☐	Phone	defaultPhone_PhoneString	YES	Equal To
20	☐	Fax	defaultFax_FaxString	NO	Equal To
21	☐	Email	defaultEmail_EmailString	NO	Equal To
22	☐	Internet Address	defaultInetAddress_InetAddressString	NO	Equal To
23	☐	Default Category	defaultCategory	NO	Equal To

Check All - Uncheck All edit

A contact is a duplicate when:

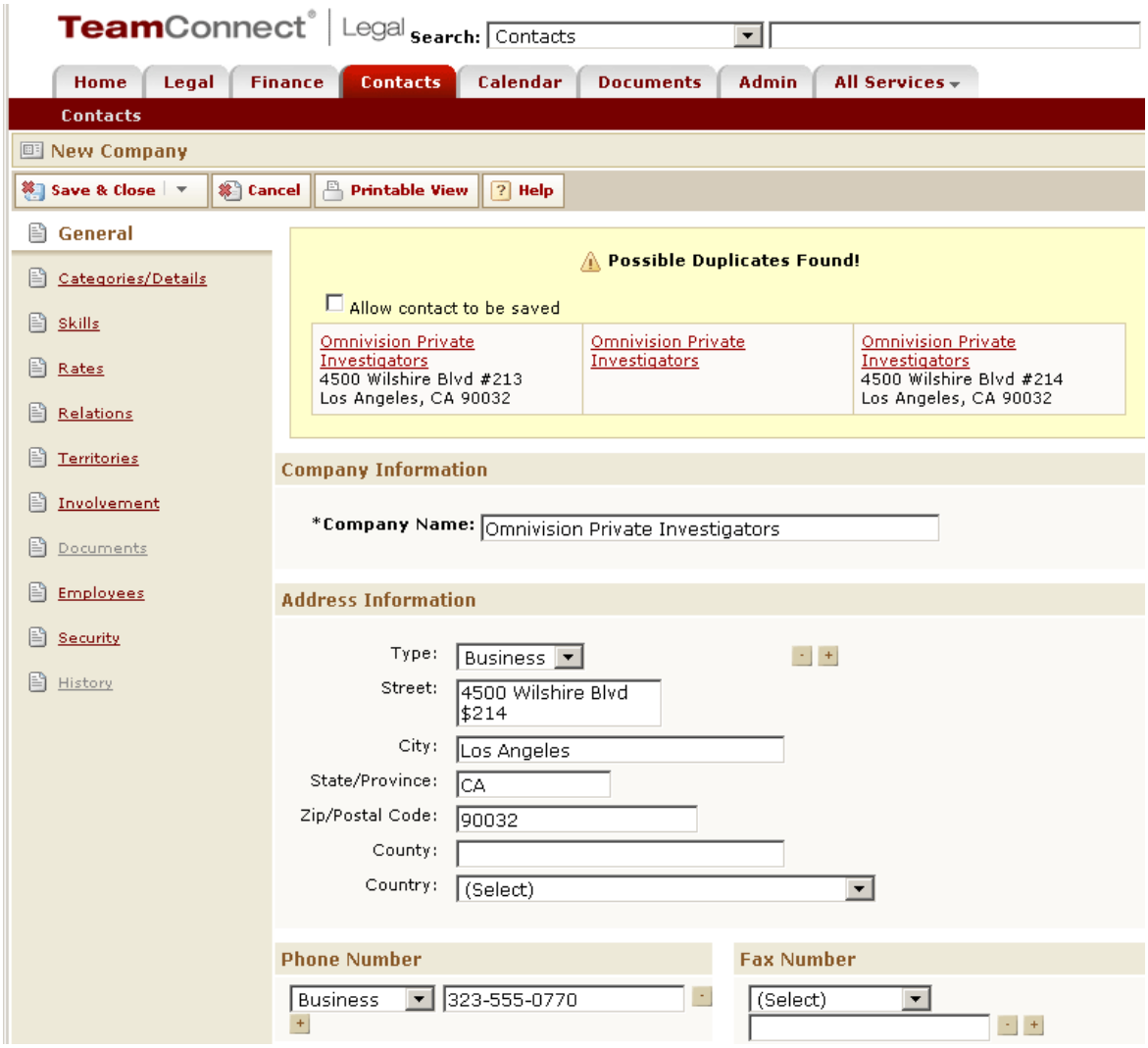
- ○ All of the above conditions are met (AND logic)
- ○ Any of the above conditions are met (OR logic)
- ◉ The following combination of the above conditions is met (2 and 4 and 19) or (2 and 4 and 16)

Duplicate Matching Settings Interface

In this example, the user may try to create a contact (type company) with the following field data:

- **Company Name** - Omnivision Private Investigators
- **Postal Code** - 90032
- **Default Phone Number** - 323-555-0770

In the same way as explained previously, when the user clicks **Save**, a message appears showing possible duplicates that have been found:



Possible Duplicate Contacts Found Message Example 2

Notice that the duplicates differ slightly from each other. In some cases, the Duplicate Contact Manager found duplications in company name and phone number. In others, it found duplications in company name and postal code. Both kinds of cases were specified in the Manager, so duplicates of both kinds are reported.

In this situation, the user may take one of the following actions:

- Override the duplicate contact check by selecting the **Allow contact to be saved** check-box and clicking **Save**. This option is only displayed if you selected the **Allow users to override** check-box on the Duplicate Contact Manager Settings screen.
- Click an existing contact link to verify that the contact is actually unique.

1.1.2.4 Using XML Worksheet Tool

The XML Worksheet Tool executes Extensible Markup Language (XML) requests that can perform functions such as inserting, updating, and deleting data in TeamConnect. This can be useful in cases where you need to create a large number of records for testing performance or search functionality, or to populate a fresh instance of TeamConnect with "stock" data.

To execute XML code using XML Worksheet Tool

1. In the Designer window, in the **Tools** drop-down list select **Xml Worksheet**.

   The corresponding **Xml Worksheet** screen appears.

2. Enter your XML code in the **Xml** text box. Alternately, click the **Choose File** button to locate a local XML document that contains the code you want to run. Click the **Upload** button after the local XML document is selected to populate the **Xml** text box with the contents of the document.

   Click the **Clear Request** link to reset the **Xml** text box to empty.

3. Click the **Post Data** button to execute the XML code in the **Xml** text box.

   The results of your XML code display in the **Results** text box.

XML Worksheet Code Format

The XML code used in XML Worksheet Tool requires several sections that must be organized in the following order:

- Request—XML document wrapper.
- Authentication—The TeamConnect username and password of an account that has the proper rights to perform the operation defined in the XML code.
- One or more records with the following components and their respective attributes:
  - **Entity** and **Operation**—Entities correspond to System Object definitions with special case for Custom Object Definitions. Operations are used to insert, delete, or update records.
  - Elements—These represent System or Custom Fields. System Field tags are generally their labels. Custom Field tags are their field names.

In TeamConnect, tags represent fields that have corresponding methods in the API. All System Objects and their system fields can be accessed with XML. Custom Objects and their fields can also be accessed with XML.

Sample XML Worksheet Code

The following XML code sample will create a contact of type Person named "John Doe" in TeamConnect. The sample contains all the sections described in XML Worksheet Code Format.

```
<TeamConnectRequest>
    <Authentication>
        <Username>TCUser</Username>
        <Password>TCPassword</Password>
    </Authentication>
    <Contact op="insert">
        <Type>P</Type>
        <FirstName>John</FirstName>
        <Name>Doe</Name>
        <Alias>Test Contact</Alias>
    </Contact>
</TeamConnectRequest>
```

1.1.2.5 Creating Custom Tools

You may develop and create custom tools to meet the specific needs of your organization using Java and XML code. For information on creating custom tools, see Using the TeamConnect API.

Points To Remember

When creating custom tools, keep in mind:

- If you grant access to a tool, the user is able to both view and use the tool.
- You may grant group rights so that specific user groups can use the tool.
- You may set up loggers to monitor tools.

Note: *Custom tools are developed using Java, and the user interface for the tools is developed using XML.*

Setting Up Custom Tools

After you have developed a custom tool, you must set it up it to work within TeamConnect by doing the following operations:

- Upload the Java and XML files to TeamConnect.
- Configure the tool in the **Administer Custom Tools** screen.
- Grant rights to solution developer users who are to test the tool.
- Test the tool on a test instance of TeamConnect before uploading it to your production system.
- After testing the tool, you must grant access to the tool on the **Tools** option of group accounts. That way, end-users in the appropriate user groups can access the tool from the **Tools** link in **All Services > Tools** drop-down list in the end-user interface.

1.1.2.5.1 Uploading Custom Tool Files

Before you can use or operate custom tools, you must first upload the code files you have written to TeamConnect.

To upload the custom tool files

1. In the end-user application, select **Documents** in the tab bar.

   The **Documents** screen appears.

2. Navigate to **Top Level > System > Tools**.
3. Create a folder for the tool (for example, **Batch_Invoicing**).

   Important: *The name of the custom tool folder cannot have spaces and it must be the same as the name of the main class file.*

4. In the new folder, create the following folders:
   - **Classes**
   - **Resource**

5. Upload the Java files to the **Classes** folder.
6. Upload the XML files to the **Resource** folder.

   After you have uploaded the necessary files, you must configure the custom tools in TeamConnect.

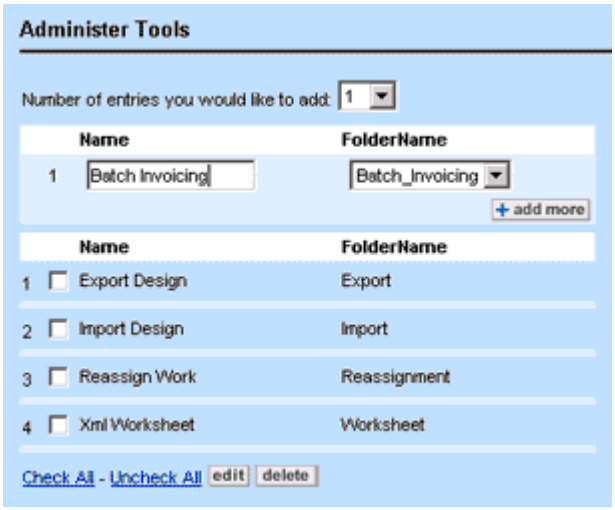
1.1.2.5.2 Configuring Custom Tools

To enable your custom tools, you must name them and point to their folder from the **Administer Tools** screen.

To configure a custom tool

1. In the Designer window, in the **Tools** drop-down list select **Administer Tools**.

   The corresponding **Administer Tools** screen appears.



Administer Tools Screen

2. Enter a name for the tool in the **Name** field.
3. From the **Folder Name** drop-down list select the directory in which the class and XML files are located.
4. Choose whether this tool will appear in the end-user interface (**User Tool**) or in the Designer interface (**Setup Tool**).
5. Click **add more**.

   The custom tool is now available in TeamConnect.

Important: *You must give rights to users on the* ***Tools*** *tab of the user or group accounts in order for them to access the custom tool.*

1.1.2.6 Appenders

Appenders are definitions of logs. They are classified according to the type of output they generate and whether they are defined for system or audit logging.

Although many features of appenders and loggers are configured in Designer, there are some features that are controlled by the **Admin** tab in the main TeamConnect user interface. These Admin features include:

- Turning loggers on and off
- Selecting the severity levels that generate log entries

When the TeamConnect application starts running, all logs defined by active appenders start to log generated messages. When you define a new appender or make changes to an existing appender, the changes take effect the next time you restart TeamConnect on the application server. When you delete an appender, the log continues to be generated until you restart TeamConnect.

You can define the following types of appenders which generate logs in corresponding output formats:

- File Appenders—Define logs that capture messages in a text file as they are generated by the system.
- SMTP Appenders—Define logs to be sent by email when a log event and when a severity of **Error** or higher is logged to a buffer.
- Socket Appenders—Define logs that transmit logger messages to a port on a remote host.

If you need a system log that only logs messages for a certain area of TeamConnect, such as custom blocks, you can associate the appender with the corresponding logger. For more details, see Loggers.

1.1.2.6.1 Default System and Audit Appenders

By default, TeamConnect has several active file appenders--that is, the logs that they generate are written as text files. These files are stored in the location specified for log folders at the time when TeamConnect was installed. There are three default logging appenders:

- **Default** (for system logging)—Logs all of the messages that are generated by all of the system loggers and writes them to the **system.log** file. You can deactivate the **Default** appender, but it cannot be deleted.

  If you want a log that isolates messages for a specific area of TeamConnect, you must define an appender for the corresponding logger.

  - For instructions on defining a system appender, see Defining System Appenders.
  - For a description of the Default system log file, see Pattern Example.
- **XML Appender**—Logs messages regarding XML requests, according to the level selected for the **XML** logger, and writes them to the **xml.log** file.
- **Default** (for audit logging)—Logs messages from all active audit loggers and writes them to the **audit.log** file. You can deactivate the **Default** appender, but it cannot be deleted.

Note: *If Collaborati Spend Management is installed on your system, the* ***Collaborati Appender*** *is installed by default.*

1.1.2.6.2 Viewing Logs Generated by File Appenders

The system and audit logs generated by file appenders are stored on the web server in a location that is specified when TeamConnect is installed. If you have access to the web server, you can directly view the logs on the server.

If you do not have access to the web server, you can view the logs directly in TeamConnect using the **View Log** hyperlink that is provided for each file appender.

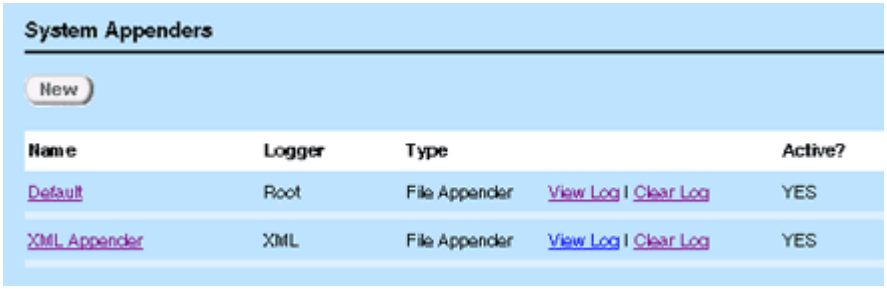
Tip: *To find out where your logs are stored, see the path displayed in the* ***Application Logs Folder*** *field in a file appender screen.*

The log location is specified using the app.logFolder parameter in the Web Application Deployment Descriptor when TeamConnect is installed.

To view a log file

1. On the **Admin** menu bar in the **System Settings** drop-down list, select either **System Logging** or **Audit Logging**, depending on the type of log you want to view.

   The **Logging** screen opens, with the corresponding tab displayed by default.



Views or Control Logs

2. In the **System Appenders** (or **Audit Appenders**) section of the tab, locate the name of the log you want to view and click the corresponding **View Log** hyperlink.

   The current log file for the log you selected is displayed in a new browser window. The most recent log events are listed at the bottom of the log file.

3. To view the latest log messages, press F5 or click your browser's **Refresh** button.

1.1.2.6.3 Clearing Logs Generated by File Appenders

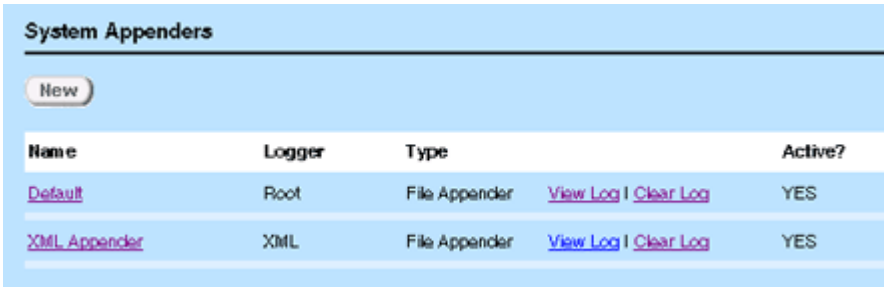
If the **Rollover Policy** field of an active file appender is set to **Rollover on schedule**, you can manually trigger the removal of the current log entries in a log file and start logging again in an empty file.

To clear a log file

1. On the **Admin** menu bar in the **System Settings** drop-down list, select either **System Logging** or **Audit Logging**, depending on the type of log you want to clear.

   The **Logging** screen opens, with the corresponding tab displayed by default.

2. In the **System Appenders** (or **Audit Appenders**) section of the tab, click the **Clear Log** hyperlink next to the file appender log you want to clear.



Clearing a Log File

   The current log entries in the log file are removed and logging begins in the empty file.

3. To view any new log messages, click the corresponding **View Log** hyperlink.

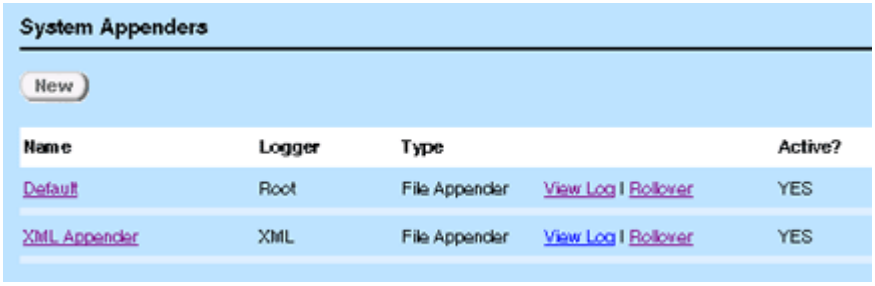
1.1.2.6.4 Rolling Over Logs Generated by File Appenders

If the **Rollover Policy** field of an active file appender is set to **Rollover on size**, you can manually trigger the rollover of the current log file and start logging again in new file.

To clear a log file

1. On the **Admin** menu bar in the **System Settings** drop-down list, select either **System Logging** or **Audit Logging**, depending on the type of log you want to clear.

   The **Logging** screen opens, with the corresponding tab displayed by default.



System Appenders Section

2. In the **System Appenders** (or **Audit Appenders**) section of the tab, click the **Rollover** hyperlink corresponding to the file appender log you want to roll over.

   The current log file is rolled over and logging begins in an empty file.

3. To view any new log messages, click the corresponding **View Log** hyperlink.

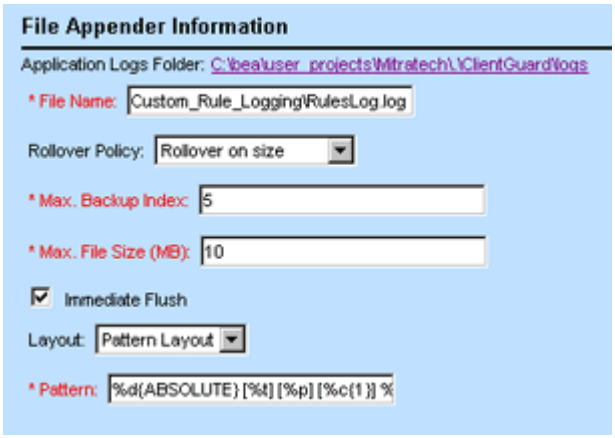
1.1.2.6.5 File Appenders

File appenders define logs that capture messages in a text file as they are generated by the system. The log file is stored in the location that was specified when TeamConnect was installed.

Whether you are defining a file appender for system logging or audit logging, the settings that are specific to file appenders are the same.

For general instructions on defining appenders, see one of the following sections:

- For system logging, see Defining System Appenders.
- For audit logging, see Defining Audit Appenders.



File Appender Information Section

The following table describes the file appender settings.

File Appender Information

Field or Button	Description
Application Logs Folder	Indicates the location of the log files on the application server as a hyperlink. This location is specified when TeamConnect is first installed. ***Note:*** *You can click the hyperlink to view the contents of the folder if you are using the same machine where TeamConnect is installed. Otherwise, navigate to the folder on the application server.*
File Name	Type the name that you want to use as the name of the log file, including its file extension. If you are using HTML or XML layout for the log file, use the corresponding extension. If you want to specify a subdirectory of the logs folder on the application server, you can type the directory location in this field, relative to the logs folder indicated in the **Application Logs Folder** field. For example, if you want the **RulesLog.log** file to be stored in a subdirectory called **Custom_Rule_Logging**, then you would type Custom_Rule_Logging\RulesLog.log as the file name. ***Important:*** *To organize your log files, you must manually create the corresponding subdirectories directly on the application server.*
Rollover Policy	Determines when you want the system to stop writing to a file, add a timestamp to the file name, and begin writing to a new file. Select

	one of the following: • **Rollover on Schedule—**Allows the log file to be rolled over daily, weekly, or monthly, as specified by the value of the **Backup Schedule** setting. ***Important:*** *If you select* ***Rollover on Schedule****, previously rolled log files are not automatically replaced by new files. You have to periodically clean up your logs directory to delete old log files.* • **Rollover on Size—**Allows you specify a file size at which the log file is rolled over and how many backup files to store before replacing them. This is the default rollover policy.
Max. Backup Index	The maximum number of backup files to be stored before backup files are replaced by new files. The oldest file is replaced each time a new file is rolled over. By default, the maximum backup index is 5. ***Important:*** *Specifying a larger number, such as 10, may negatively impact performance.* This field is only displayed if **Rollover on Size** is selected as the rollover policy.
Max. File Size (MB)	The maximum file size, in megabytes, that the log file should reach before rolling over and starting a new file. By default, the maximum file size is 10 MB. This field is only displayed if **Rollover on Size** is selected as the rollover policy.
Backup Schedule	Select how often the log file should be rolled over and timestamped and a new file should be created—**Daily**, **Weekly**, or **Monthly**. This field is only displayed if **Rollover on Schedule** is selected for the rollover policy.
Immediate Flush	Select this check-box to have each write of a logging event followed by a flush operation. This helps prevent all or part of a log message from being lost should the application exit abruptly. This check-box is selected by default. If this check-box is not selected, logging throughput is improved by about 15%, but it is more likely that if the application exits abruptly, a logging message that was written might not be completely captured in the log.

Layout	Select whether you want to use a **Pattern**, **HTML**, or **XML** layout for the contents being written to the log file. **Pattern Layout** is selected by default. For more details, see File Appender Layouts.
Pattern	This field becomes available when **Pattern Layout** is selected in the **Layout** field. Type the pattern you want to use to display the logging messages in the log file. For more details, see Pattern Layout.

1.1.2.6.5.1 File Appender Layouts

When you define a file appender, you must specify how you want the contents of the file to be organized and formatted. Log4j® provides several types of layouts for file appenders, including Pattern, HTML, and XML layouts. Each of these layouts has the following advantages and disadvantages:

Comparison of File Appender Layouts

Layout	Advantages	Disadvantages
Pattern Layout	Provides easy control over content and organization of log file	• Limited control over output • Difficult to parse for specific information
HTML Layout	• Requires no configuration or formatting • Output is clean, organized and easy to read	• Fixed format • Difficult to parse for specific information
XML Layout	Easiest output to parse for specific information using other software	• Might require more configuration and design files to be created and implemented • More difficult to implement

The pattern layout is a flexible layout approach for file appenders. You specify what information you want to include in the log file by providing a string of characters in the **Pattern** field of a file appender. These instructions are interpreted by the system, so that the resulting log file contains the information you need.

The pattern for a file appender is constructed using conversion specifiers and static characters you might want to include, as demonstrated in the Pattern Example.

Conversion Specifiers

Conversion specifiers indicate what information to write to the log. The following table describes all possible conversion specifiers for pattern layouts.

Log4j® Pattern Layout Conversion Specifiers

Conversion specifier	Output in log file
%c	The name of the logger with which the logging event is associated. If desired, you can specify the desired number of right-most components of the logger name to be printed in the log by following the %c with a precision specifier (a decimal constant within braces). By default, the logger name is printed in full. For an example of how the Default system log uses a precision specifier with this conversion specifier, see Pattern Example.
%C	The fully qualified class name of the calling object issuing the logging request. If desired, you can specify the desired number of right-most components of the class name to be printed in the log by following the %C with a precision specifier (a decimal constant within braces). See Pattern Example for an example of how the Default system log uses a precision specifier with this conversion specifier. ***Caution:*** *Generating this information for a log is extremely slow.*
%d	The date of the logging event. If desired, you can follow the %d with a date format specifier within braces. For example, %d{HH:mm:ss,SSS} and %d{dd MMM yyyy HH:mm:ss} are acceptable. By default, ISO8601 format is used. Log4j® has its own date formatters. These can be specified using one of the following: • ABSOLUTE • DATE • ISO8601 See the Pattern Example for an example of how the Default system log uses a date format specifier with this conversion specifier.
%F	The file name where the logging request was issued. ***Caution:*** *Generating this information for the log is extremely slow.*

%l	The location information of the calling object which generated the logging event. Location information includes the file name where the logging request was issued, the line number in the file, the class name, and the method name from which the message was logged. ***Caution:*** *Generating this information for the log is extremely slow.*
%L	The line number of the file where the logging request was issued. ***Caution:*** *Generating this information for the log is extremely slow.*
%m	The message associated with the logging event.
%M	The method name where the logging request was issued. ***Caution:*** *Generating this information for the log is extremely slow.*
%n	The platform-dependent line separator character or characters.
%p	The level of the logging event.
%r	The number of milliseconds elapsed from the start of TeamConnect until the creation of the logging event.
%t	The name of the thread that generated the logging event.
%%	A single percent sign.

Pattern Example

The following pattern shows the use of several conversion specifiers that write information to the log. This is the pattern used by the **Default** system log:

```
%d{ABSOLUTE} [%t] [%p] [%c{1}] %C{1}.%M - %m%n
```

This can be roughly translated as the following:

```
date [thread that generated the logging event] [level of the logging event] [logger
of the logging event] class.method - message followed by a line return
```

For example, an entry in the Default system log might be:

```
11:20:44,005 [jessica] [DEBUG] [sql] TCLog$SQLLogWriter.write - UnitOfWork(2420720)
```
—end unit of work commit

Notice that the spaces, brackets, hyphen and period in this example are printed in the generated line of text in the log as static text.

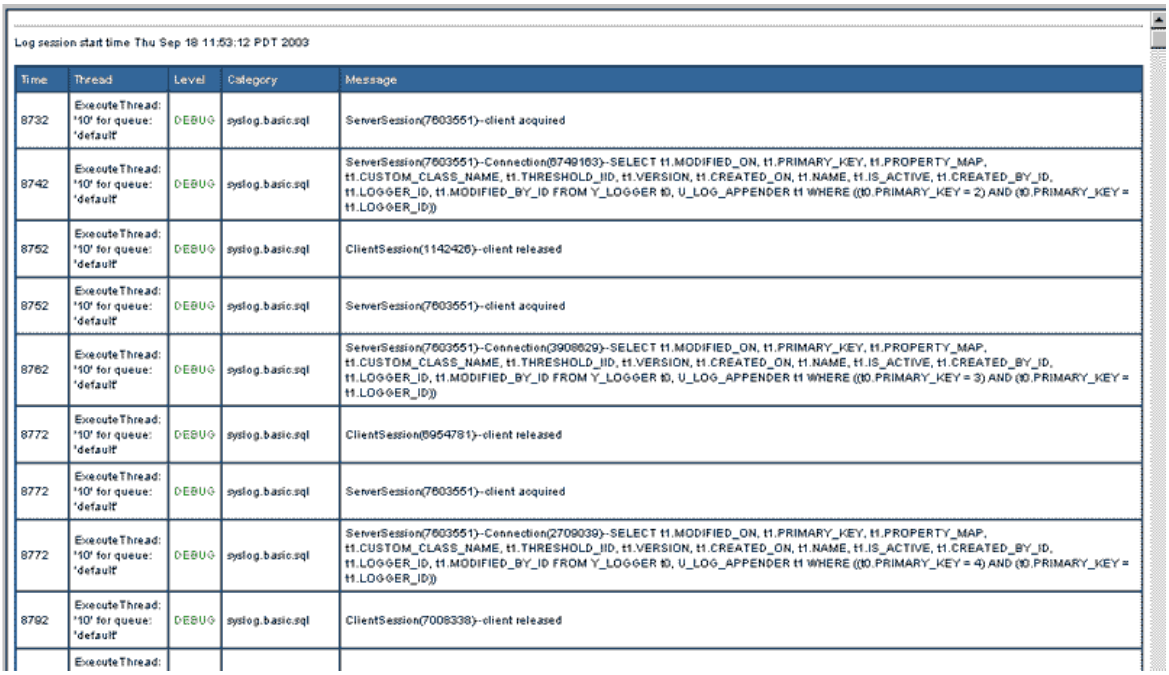
The HTML layout for file appenders writes logging events in a clean-looking table. Each logging event written to the log is added as a row in the table. The columns in this table are described as follows:

File Appender HTML Layout Columns

Time	Thread	Level	Category	Message
Milliseconds elapsed since the TeamConnect was started until	Name of the thread that generated the event	Level of the event	Logger associated with the event	Message associated with the event

Tip: *In HTML layout, the time of an event is displayed relative to the start time of the application. To track events logged over long periods of time, specify frequent roll over.*

The following table is an example of a log file in HTML layout:



Log session start time Thu Sep 18 11:53:12 PDT 2003

Time	Thread	Level	Category	Message
8732	ExecuteThread: '10' for queue: 'default'	DEBUG	syslog.basic.sql	ServerSession(7603551)--client acquired
8742	ExecuteThread: '10' for queue: 'default'	DEBUG	syslog.basic.sql	ServerSession(7603551)--Connection(8749163)--SELECT t1.MODIFIED_ON, t1.PRIMARY_KEY, t1.PROPERTY_MAP, t1.CUSTOM_CLASS_NAME, t1.THRESHOLD_IID, t1.VERSION, t1.CREATED_ON, t1.NAME, t1.IS_ACTIVE, t1.CREATED_BY_ID, t1.LOGGER_ID, t1.MODIFIED_BY_ID FROM Y_LOGGER t0, U_LOG_APPENDER t1 WHERE ((t0.PRIMARY_KEY = 2) AND (t0.PRIMARY_KEY = t1.LOGGER_ID))
8752	ExecuteThread: '10' for queue: 'default'	DEBUG	syslog.basic.sql	ClientSession(1142426)--client released
8752	ExecuteThread: '10' for queue: 'default'	DEBUG	syslog.basic.sql	ServerSession(7603551)--client acquired
8762	ExecuteThread: '10' for queue: 'default'	DEBUG	syslog.basic.sql	ServerSession(7603551)--Connection(3908629)--SELECT t1.MODIFIED_ON, t1.PRIMARY_KEY, t1.PROPERTY_MAP, t1.CUSTOM_CLASS_NAME, t1.THRESHOLD_IID, t1.VERSION, t1.CREATED_ON, t1.NAME, t1.IS_ACTIVE, t1.CREATED_BY_ID, t1.LOGGER_ID, t1.MODIFIED_BY_ID FROM Y_LOGGER t0, U_LOG_APPENDER t1 WHERE ((t0.PRIMARY_KEY = 3) AND (t0.PRIMARY_KEY = t1.LOGGER_ID))
8772	ExecuteThread: '10' for queue: 'default'	DEBUG	syslog.basic.sql	ClientSession(5954781)--client released
8772	ExecuteThread: '10' for queue: 'default'	DEBUG	syslog.basic.sql	ServerSession(7603551)--client acquired
8772	ExecuteThread: '10' for queue: 'default'	DEBUG	syslog.basic.sql	ServerSession(7603551)--Connection(2709039)--SELECT t1.MODIFIED_ON, t1.PRIMARY_KEY, t1.PROPERTY_MAP, t1.CUSTOM_CLASS_NAME, t1.THRESHOLD_IID, t1.VERSION, t1.CREATED_ON, t1.NAME, t1.IS_ACTIVE, t1.CREATED_BY_ID, t1.LOGGER_ID, t1.MODIFIED_BY_ID FROM Y_LOGGER t0, U_LOG_APPENDER t1 WHERE ((t0.PRIMARY_KEY = 4) AND (t0.PRIMARY_KEY = t1.LOGGER_ID))
8792	ExecuteThread: '10' for queue: 'default'	DEBUG	syslog.basic.sql	ClientSession(7008338)--client released
	ExecuteThread:			

File Appender HTML Layout Example

The XML layout writes logging events in a fixed format. This layout is implemented without more configuration for TeamConnect. Therefore, it does not generate a completely well-formed XML file. It is designed to be included as an external entity in a separate file to form a correct XML file. This allows you to determine how the logging events should be displayed in the log.

The XML layout can be useful for creating output that can be parsed by other software to create a report of specific information. For example, to create an audit report containing a summary of methods accessed by users, you can more easily parse XML than pattern text or HTML.

The following is an example output of a logging event in XML layout:

```
<log4j:event logger="syslog.basic.sql" timestamp="1063916251658" level="DEBUG"
thread="ExecuteThread: '10' for queue: 'default'">
<log4j:message><![CDATA[ServerSession(4688271)--client acquired
]]></log4j:message>
</log4j:event>
```

To utilize the XML layout, you need to create the necessary files to use the XML output as content that is pulled in and formatted correctly.

1.1.2.6.6 SMTP Appenders

TeamConnect's SMTP appender collects the logger messages in a buffer with a specified size. When the buffer receives a message with a severity of **Error** or higher, the SMTP appender sends an email message with the log events currently in the buffer to the specified email addresses.

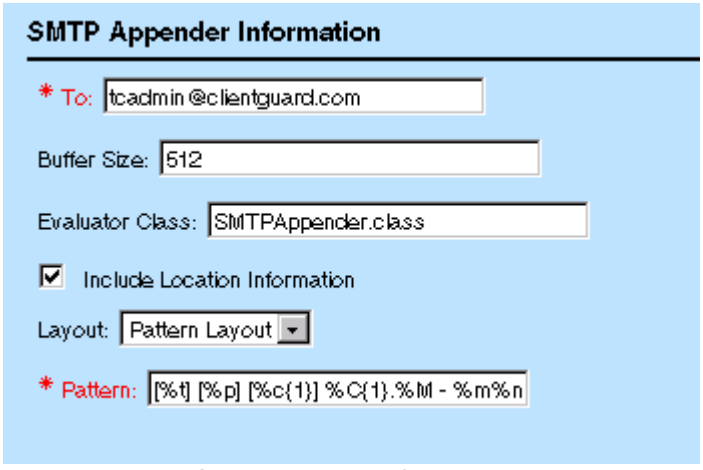
To display the SMTP Appender Information

1. On the Admin menu in the **System Settings** drop-down list, select **System Logging** or **Audit Logging**, depending on which type of logging you need to define.
2. In the **System Appenders** section, click **New**.

   A new appender screen opens.
3. Select **SMTP Appender** from the **Appender Type** drop-down list.

   The **SMTP Appender Information** section is displayed.

For information on how to define appenders, see Defining System Appenders or Defining Audit Appenders. Whether you are defining an SMTP appender for system or audit logging, the SMTP appender settings are the same. The following image shows a set of sample settings:



SMTP Appender Information

In addition to the fields described in the following table, you must configure the **Outgoing Mail Server Settings** section of the **Email** page under **Admin Settings**. For more details, see the Admin Settings, Email Page Field Descriptions table.

SMTP Appender Information

Field	Description
To	The email addresses to which the log should be sent when a triggering log event occurs. Use commas to separate each address.
Buffer Size	The maximum size, in kilobytes, of the buffer that collects logging events. When the maximum buffer size is reached, the oldest events are deleted as new events are added to the buffer. The default buffer size is 512 kilobytes.
Evaluator Class	To specify a log level other than Error (the default) to trigger email messages, you must upload a class that modifies the triggering level. Enter the name of the class file in this field.
Include Location Information	Select this option to include the location information in the logging events placed in the buffer. For a description of the location information, see the Log4j® Pattern Layout Conversion Specifiers table. If this option is not selected, the location information is replaced with a question mark (?) in the log. However, it can help reduce the generation time of the logging events.
Layout	Select whether you want to use a **Pattern**, **HTML**, or **XML** layout for the contents being written to the buffer. **Pattern Layout** is selected by default. The layout functionality is identical to that of the file appender. For more details, see File Appender Layouts.
Pattern	This field becomes available when **Pattern Layout** is selected in the **Layout** field. Type the pattern you want to use to display the logging messages in the log file. For more details, see Pattern Layout.

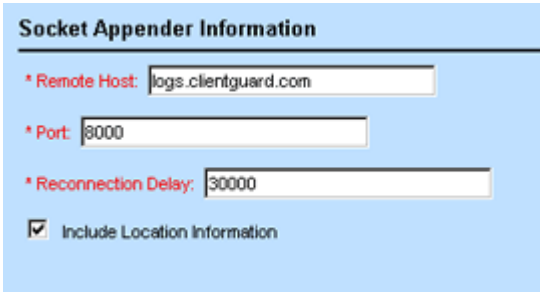
1.1.2.6.7 Socket Appenders

A socket appender transmits logger messages to a port on a remote host. On the remote host, the messages can then be logged as if generated locally. Using a socket appender allows you to send

logs from various systems to a single server where all logs for your organization are monitored or stored.

To use a socket appender, you must define a socket server to receive the log messages. Log4j® includes socket server examples as part of its distribution.

Whether you are defining a socket appender for system logging or audit logging, the settings shown in the following image are the same.



Socket Appender Information

The following table describes the fields in the **Socket Appender Information** section of the Appender screen.

Socket Appender Information

Field	Description
Remote Host	The name or IP address of the server to which the logging events should be sent.
Port	The port number on the remote host to which the logging events should be sent.
Reconnection Delay	The number of milliseconds to wait between each failed attempt to connect to the server using the specified port. The default is 30000 milliseconds (30 seconds). If you set this field to zero (0), the socket appender attempts to connect to the server only once. If the connection fails, TeamConnect does not try to reconnect.
Include Location Information	Select this option to include the location information in the logging events placed in the buffer. Location information includes the file name where the logging request was issued, the line number in the file, the class name, and the method name from which the message was logged. If this option is not selected, then the location information is replaced with a question mark (?) in the log. However, it can help reduce the generation time of the logging events.

For general instructions on defining all types of appenders, see one of the following:

- For system logging, see Defining System Appenders.
- For audit logging, see Defining Audit Appenders.

1.1.2.6.8 Defining System Appenders

Although several default system appenders are provided, you can define new system appenders as needed. When you define a system appender, you are defining a log that you want generated for either one specific system logger or for all of the system loggers.

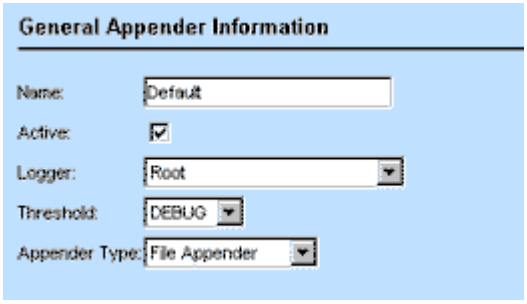
For example, you might want to define a log that only captures messages regarding custom rules that are defined for your implementation of TeamConnect. You can do this by defining an appender for the **Rule** logger.

Note: *For descriptions of the system loggers, see System Loggers.*

To define a system appender

1. On the **Admin** menu in the **System Settings** drop-down list, select **System Logging**.
2. If the **Audit** tab is displayed in the **Logging** screen, select the **System** tab.
3. In the **System Appenders** section, click **New**.

   A new appender screen opens.



General Appender Information for System Logging

4. In the **General Appender Information** section, enter the desired information as described in the General Appender Information for System Logging table.
5. Depending on which type of appender you selected in the **Appender Type** field, enter the desired information in the specific fields for file, SMTP, or socket appenders as described in the corresponding tables:
   - The File Appender Information table.
   - The SMTP Appender Information table.
   - The Socket Appender Information table.
6. Click **Save**.

7. Restart the TeamConnect application on the application server.

The general information that must be filled out for all system appenders, regardless of their type, is shown in the General Appender Information for System Logging image.

The following table describes the fields in the **General Appender Information** section:

General Appender Information for System Logging

Field	Description
Name	The name of the system appender as you want it to be identified in the list of system appenders in the **System Logging** screen.
Active	Select this check-box to activate the appender. Clear to deactivate the appender. Your change takes effect the next time the TeamConnect application is restarted.
Logger	Select whether you want the system appender to be defined for all loggers or one specific logger. The appender logs all messages associated with the logger you select. If you want the appender to be defined for all loggers, select **Root**. For more details about the system loggers, see System Loggers.
Threshold	Select the threshold for the system appender. Allows you to limit the severity of messages to be captured in a particular log. The threshold is appender-specific, whereas the logger levels themselves are system-wide. For example, if you want a certain log to capture only Error and Fatal messages, but you want another log to capture all levels of messages, you can define an appender with the threshold set to **Error**, even though the logger level itself is set to **Debug**.
Appender Type	Select the type of appender you want to define. For details about each appender type, see the following: • File Appenders • SMTP Appenders • Socket Appenders

1.1.2.6.9 Defining Audit Appenders

Appenders for audit logging differ from system appenders in the following ways:

- Because you do not have to set the levels of the audit loggers, your audit log appender does not need a threshold for which levels to log.

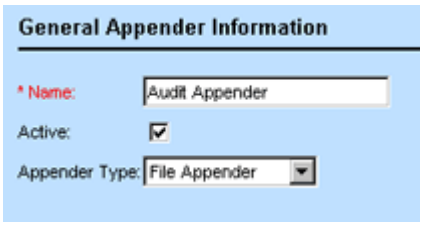
- You cannot define an appender for a single audit logger. Messages from all audit loggers that are turned on are logged in an audit appender.

For descriptions of the audit loggers, see Audit Loggers.

To define an audit appender

1. On the **Admin** menu in the **System Settings** drop-down list, select **Audit Logging**.
2. If the System tab is displayed in the **Logging** screen, select the **Audit** tab.
3. In the **Audit Appenders** section, click **New**.

   A new appender screen is displayed.



General Appender Information for Audit Logging

4. In the **General Appender Information** section, enter the desired information as described in the General Appender Information for Audit Logging table.
5. Depending on which type of appender you selected in the **Appender Type** field, enter the desired information in the specific fields for file, SMTP, or socket appenders as described in the corresponding table:
   - The File Appender Information table.
   - The SMTP Appender Information table.
   - The Socket Appender Information table.
6. Click **Save**.
7. Restart the TeamConnect application on the application server.

After you restart TeamConnect, messages are written to the log that you defined.

The following table describes the general appender information that must be filled out for all audit appenders, regardless of their type.

General Appender Information for Audit Logging

Field	Description
Name	The name of the audit appender as you want it to be identified in the list of audit appenders in the **Audit Logging** screen.
Active	Select this check-box to activate the appender. Clear to deactivate the appender.

	Your change takes effect the next time the TeamConnect application is restarted.
Appender Type	Select the type of appender you want to define. For details about each appender type, see the following: • File Appenders • SMTP Appenders • Socket Appenders

1.1.2.6.10 Loggers

Loggers categorize the messages that can be captured in a log according to the area of TeamConnect where that message is generated. Loggers are predefined according to area. You cannot create new loggers.

When you need to log messages from a certain area of TeamConnect, you define an appender for the logger that is associated with that area. For example, you can create a log for the portal panes area by defining an appender for the **Portal** logger. You can define a log for the user interface components by defining an appender for the **UI Components** logger.

This section includes the following topics:

- System Loggers
- Audit Loggers

1.1.2.6.10.1 System Loggers

System loggers categorize messages in the code according to the area of TeamConnect that generates them, so that you can specify which area's messages you want to capture in a particular log. You can define multiple logs that capture messages from different TeamConnect components or all TeamConnect components.

For each log that you have defined, you can specify an overall threshold for what severity of messages you want written to it, so that the logs contain only the most pertinent information. For more details about the threshold for a log (or appender), see the General Appender Information for System Loggers table.

For each available TeamConnect component, you can specify what severity (level) of messages you want written to the logs regarding this component, so that the logs contain only the most pertinent information about the component.

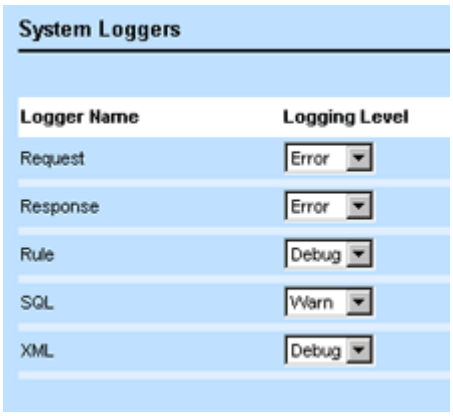
The loggers that are available for system logging are listed on the **System** tab of the **Logging** screen, as shown in the System Loggers and Detailed System Loggers images.

All of the system loggers have **Root** as their parent logger. This hierarchy allows appenders defined for the Root logger to be available to descendant loggers. For example, this means that the **Default** appender, which has been defined for the Root logger, receives logging statements from all of the loggers in the hierarchy.

Tip: *Because the Default appender captures messages from all loggers in the system, you do not need to define another appender for the Root logger.*

You can define an appender to be the target of a single logger in the hierarchy. This appender only receives logging statements from the logger for which it is defined. For example, if you need a text file that logs events for the batch display component, you define an appender for the **Batch Display Component** logger.

The following image shows the **System Loggers** section of the **Logging** screen:



System Loggers

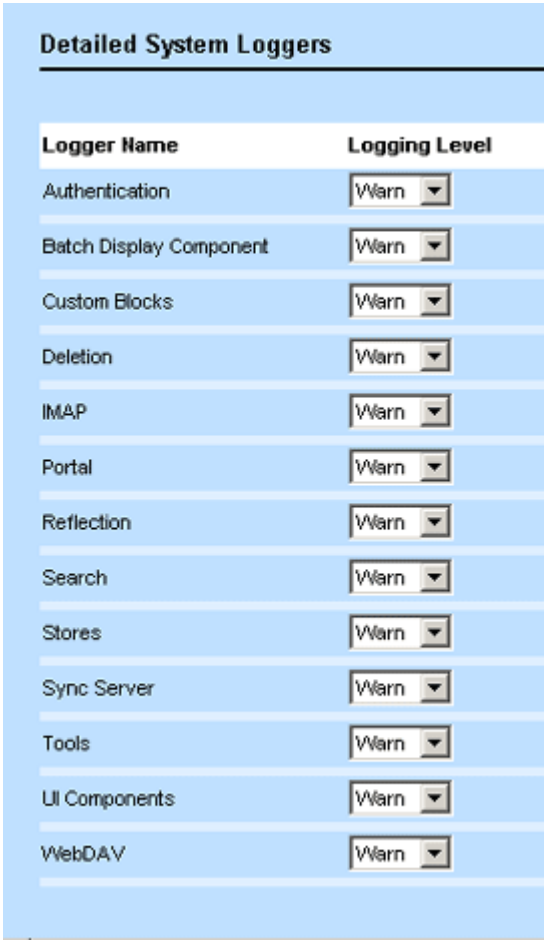
The following table describes the **System Loggers** section of the **Logging** screen:

System Loggers Table

Logger	Description
Request	Logs HTTP request information including headers, cookies, and parameters. This logger can automatically be set to the Debug level each time TeamConnect application is restarted. This depends on the value of the syslog.basic.request parameter in the Web Application Deployment Descriptor.
Response	Logs HTTP response header information.
Rule	Logs messages generated by rules that have been defined for your implementation of TeamConnect.
SQL	Logs EclipseLink-generated SQL statements. This logger can automatically be set to the Debug level each time TeamConnect application is restarted. This depends on the value of the syslog.basic.sql parameter in the Web Application Deployment Descriptor. For more details about this parameter and other web application parameters, see the Installation Guide.

XML	Logs messages regarding XML requests.

The following image shows the **Detailed System Loggers** section of the **Logging** screen:



Detailed System Loggers

The following table describes the Detailed System Loggers section of the Logging screen:

Detailed System Loggers Table

Logger	Description
Authentication	Logs all messages regarding user authentication.
Batch Display Component	Logs all messages regarding the batch display component.
Custom Blocks	Logs all messages regarding custom blocks.
Deletion	Logs all messages regarding record deletion.

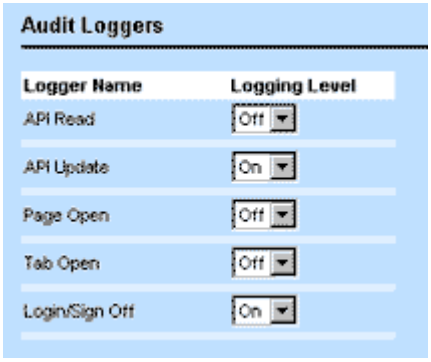
IMAP	Logs events related to the TeamConnect IMAP Server and Email Viewer.
Portal	Logs all messages regarding portal panes, including custom content.
Reflection	Logs all messages generated by TeamConnect's reflection factory.
Search	Logs all messages regarding search views and searching for records.
Stores	Logs all messages regarding TeamConnect stores.
Sync Server	Logs all messages regarding cache synchronization.
Tools	Logs all messages regarding both system and custom tools.
UI Components	Logs all messages regarding rendering of TeamConnect screens.
WebDAV	Logs all messages related to WebDAV document management.

1.1.2.6.10.2 Audit Loggers

Audit loggers send their messages to logs defined by audit appenders. Unlike system loggers, audit loggers have the following characteristics:

- Audit loggers are turned on or off system-wide. You do not have to set levels for audit loggers.
- When you define an appender for audit logging, all audit loggers that are turned on send their messages to the log. You cannot define separate logs for the various audit loggers.
- When you define an appender for audit logging, you do not have to specify a threshold. For details about defining audit appenders, see Defining Audit Appenders.

The following image shows the available audit loggers.



Audit Loggers

The following table describes the **Audit Loggers** section:

Audit Loggers Table

Logger	Description
API Read	Turn on this logger to log read calls to the API.
API Update	Turn on this logger to log update calls to the API.
Page Open	Turn on this logger to log the opening of a page by a user.
Tab Open	Turn on this logger to log the clicking of a tab by a user.
Login/Sign Off	Turn on this logger to log user sign on/off events.

1.1.3 About Objects

Objects in TeamConnect represent business entities and concepts. Each object has an object definition that determines how object *records* that store your organization's information are created, displayed, and maintained.

When you customize TeamConnect, most of your work involves modifying or creating object definitions to meet the needs of your organization. For example, you might modify the Account *system object* that comes with TeamConnect or create a new "Property" *custom object*.

Object definitions determine the basic appearance of the records created by end users, each record's name and number, required fields, workflow, and so on. You can also create object definitions that dynamically pre-populate the record with certain data or automatically create any necessary related records.

Based on your business requirements, you can specify the following aspects of object definitions:

- **Properties**—Visual and functional aspects, such as custom fields and rules that require them to be populated before users can save the record. For example, you could create the "Case Caption" custom field for the Dispute object. Then, you could create a rule so that Dispute records cannot be saved if this field is empty.
- **Relationships**—Relationships to other objects, such as a dispute record that has several involved parties, which are related to contact records.

1.1.3.1 System Objects

System objects represent business objects that are common to most businesses. TeamConnect comes with the following system objects:

- **Account**—Used to allocate and track various financial transactions.

- **Appointment**—Used to schedule events, engagements, or meetings for a particular purpose, place, and time with specific attendees (users) and resources.
- **Contact**—Information about people and companies, including rates charged for services, relations to projects and other contacts, and so on. You must create a contact record for a user before you can create their user account.
- **Contact Group**—Allows users to organize contact records in various Designer contact groups, which are called "Address Books" in the end-user interface.
- **Document**—Allows users to upload and manage files "attached" to records or directly uploaded to TeamConnect's **Documents** area. Documents are automatically available as a system block in all object views.

  You can also upload files used to customize TeamConnect, such as XML and CLASS files used for custom blocks, custom actions, and so on.
- **Expense**—Allows users to track internal costs of doing business in their organization. Expenses are available as a system block in object views for custom objects.
- **Group**—Allows administrators to define access rights, home pages, and object views per user group and assign users to one or more groups.
- **History**—Allows users to make chronological entries and track events associated with a specific record. History is a related object; that is, history records can only exist when related to other records. History is available as a system block in object views.
- **Invoice**—Allows end users to track invoices or bills sent by vendors who provide their organization with goods or services. Invoices are available as a system block in object views for custom objects.
- **Legal Setting**—Used to hold settings for TeamConnect Legal Matter Management and the budget settings records after the settings for budgets for specific years.
- **Line Item** (Invoice)—Allows end users to list the goods or services specified in a vendor invoice. Users must enter the details of each line item, such as the type (fee or expense), price, quantity, and dates in the line items screen of the corresponding invoice. Line items are available as a system block in object views for custom objects.

  Some of the object definition properties of Line Item, such as how to customize the display of specific fields, are controlled by System Setting screens. For more information, see Setting up Invoice Line Items.
- **Involved**—Allows end users to keep track of various involved parties associated with a Project. An Involved record is automatically available as a system Block in Object Views for Custom Objects. This System Object definition is always related to a Custom Object.
- **Task**—Allows end users to track internal assignments they have to do, usually while working on Projects, for example preparing reports, following up on cases, reviewing documentation, and so on. Tasks are automatically available as a system Block in Object Views for Custom Objects.
- **User**—An account created for a specific person who uses TeamConnect. Users can be members of Groups, and can have Rights assigned to them or from a group in which they are a member. Each user has their own password to log in to TeamConnect, which is first specified in their user account (accessible by the Administrator only) together with the username and rights.

The corresponding object records are available in the end-user interface—except for user and group account records, which are only available to users with administrative rights.

The Involved object record is available only through the parent custom object records. For details, see Unique Custom Object Types.

You can customize certain properties of system objects, such as their appearance, custom fields, and search views. However, you cannot modify the Group and User administrative system objects.

You cannot delete system object definitions or create new ones--except for Involved whose object definitions can be deleted from their parent custom object definitions.

System objects are represented by their own main tables in TeamConnect's object model, such as **TAccount** or **TTask**. The exceptions are User and Group objects, which are represented by the **YUser** and **YGroup** tables respectively. For details, see Object Model: Read This First plus the additional reference tables mentioned in that topic.

1.1.3.2 Custom Objects

You can create and define custom objects to meet the business requirements of your organization. For example, you might create a "Policy" object so that end users can create policy records to track each policy in your organization.

When you create a custom object, TeamConnect automatically creates an Involved object for it. For details, see Unique Custom Object Types.

You can also delete custom object definitions during the customization process. However, once your TeamConnect system is in production, deleting custom objects or their components results in permanent loss of the data associated with them.

Custom object records are often referred to as "projects" within this documentation. After you define custom objects, they appear in the user interface with the names that you specify.

Custom objects are represented in TeamConnect's object model by the **TProject** table and a number of "administrative" tables such as **WObjdProjInfo**. For details, see Object Model: Read This First plus the additional reference tables this reference points you to.

1.1.3.3 Unique Custom Object Types

The unique custom object type in TeamConnect is **Involved**.

Involved records allow users to store information about involved parties (people or companies) who are involved in a project. Involved parties have roles such as witness, outside counsel firm, opposing party, and so on.

This objects is unique because its system object definition can only exist within custom object definitions. Likewise, these records can be viewed and edited only from within a custom object record.

1.1.3.4 Related Objects

Object records are often associated or related to other records. For example, each dispute record may have multiple related account records. Objects that have relationships with other objects are referred to as related objects.

End users can access related records from hyperlinks, pages, or blocks within a record to which they are related. For example, by default, each custom object is related to certain system objects. These system objects can be accessed from the following pages inside a custom object record:

- Accounts
- Appointments
- Documents
- Expenses
- Histories
- Involved
- Tasks

1.1.3.4.1 Relationship Types

Objects in TeamConnect can have the following types of relationships:

- **Between system objects**—Most system objects are associated with the History and Document system objects. The exceptions are the objects themselves, the Contact Group (Address Book) object, and Group and User account objects.
- **Between system and custom objects**—All custom objects are associated with certain system objects, such as Account, Appointment, Expense, Task, Document, and History objects.
- **Between custom objects**—You can add custom fields of type Custom Object to custom object definitions so that end users can establish relationships between custom object records. For details, see Custom Object Field Type.
- **Contact-centric relationships**—Certain custom objects are centered on a specific contact. For details, see Contact-Centric Custom Objects.
- **Parent-child relations**—Parent (or "main") objects can have multiple child objects. Each child can have only one parent object. For example, a dispute record can have multiple negotiation records, but each negotiation can have only one dispute. This type of relationship is possible between custom object definitions and between account records. For more details, see Parent-Child Relations between Custom Objects.

  Parent-child relationships also exist between custom objects and their Involved objects, and between custom objects and embedded custom objects.

The following table lists objects and their default relationships in TeamConnect.

Related Object List

Object	Related Objects											
	Account	Appointment	Contact	Expense	History	Invoice	Line Item	Involved	Task	Document	Custom object	Embedded custom object
Account	x				x					x		

Appointment					x					x		
Contact			x		x		x			x		
Contact Group (Address Book)			x									
Document					x							
Expense					x					x		
Group					x							
History										x		
Invoice					x		x			x		
(Invoice) Line Item			x			x					x	
Involved	x				x					x		
Task					x					x		
User			x		x							
Custom object	x	x		x	x			x	x	x	x	x

Embedded custom object												

Points To Remember

In summary, *related objects* refer to:

- Child objects.
- Embedded objects
- Involved objects.
- Custom and system objects related to each other by default or through custom fields of type Custom Object.

The records of related objects can be accessed both independently from **All Services** (except for Involved, History, and embedded objects), and from hyperlinks, tabs, or blocks within the objects to which they are related.

1.1.3.4.2 Parent-Child Relations betw een Custom Objects

Because of their nature in the business world, certain custom objects cannot exist independently and always require a parent. For example, in the insurance industry, claims can only be filed against existing policies.

You can establish a parent-child relationship in the following ways:

- Create an embedded custom object within a custom object. For details, see Embedded Custom Objects.
- Create two custom objects and specify one to use the other as a parent.

  On the **General** tab of the child object definition, you select the parent object from a drop-down list and select a check box to confirm the object requires a parent. For details, see General Custom Object Information.

Results in End-User interface

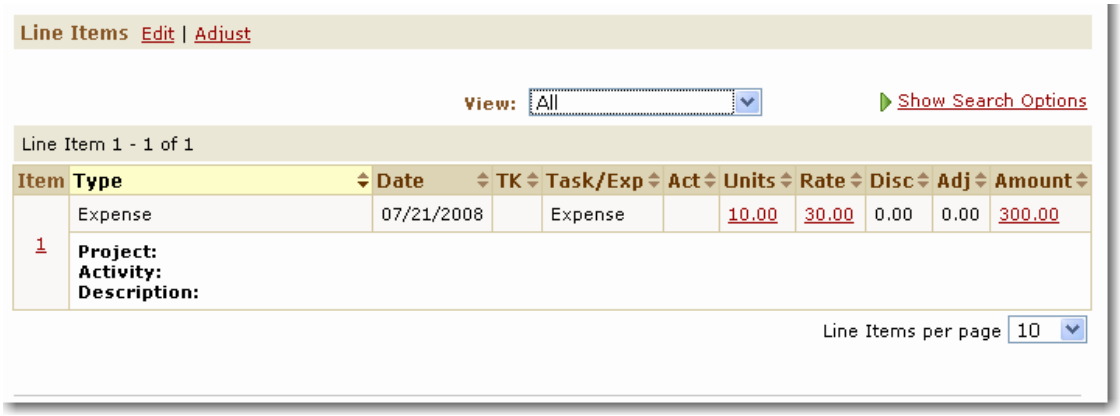
Whenever you specify a parent custom object (for example, Policy for Claim) in a custom object definition, the relationship is represented in the following areas of the end-user interface:

- **General block of child records**—A field labeled **Parent *objectName*** is automatically created. With this field, the user can select the appropriate parent record. The record cannot be saved until a parent record is selected.

  As soon as the user selects a parent record and saves the child record, the field displays the name of the selected parent record as a hyperlink, for example:

  Parent Specialized Invoice: Correli Industries Specialized Invoice

- **Parent record screens**—A new block labeled with the plural form of the child custom object, for example, **Line Items**, is created, as shown in the following image. This block has a search field and provides quick access to the child records.



Example: Child Records Object in End-User Interface

This block is automatically added to the system view of the parent object. However, if you are planning to create your own custom view for the parent object, you have to add this block to the object view manually. For details on object views, see Object Views.

- **All Services**—When you define parent and child objects, you can check this box to hide them on the **All Services** drop-down list and the **All Services** page. If this box is checked, the object is also excluded from lists in the Global Search area.

  If you choose to omit child objects from the **Go to** and **Create a new** drop-down lists, end users must go through the respective records of the parent object definition to create or access child custom object records.

  Important: *You must assign user group rights to both parent and child custom objects. Otherwise, users are not able to work with them.*

Results in Designer User Interface

Whenever you specify a parent custom object (for example, Tort for Dispute), the following changes happen in the object definition screens in the Designer interface:

- **Object Views tab**—A system block named after the child object in the plural is automatically created and become available for adding to the custom object views of the parent object.
- **Object Navigator window**—The **parent->** attribute appears in all of the child object definition's **TProject** tables, and you can use it to build attribute paths. For more details, see Using Object Navigator.

1.1.3.4.3 Contact-Centric Custom Objects

Certain custom objects are centered on one specific Contact record. Such custom objects have contact-centric relationships.

For example, a loan cannot exist without an applicant. In this case, you need all Loan records to be linked to the Contact record of the individual or company applying for it. You can ensure this relationship when you create a Loan custom object. When you configure the object's general

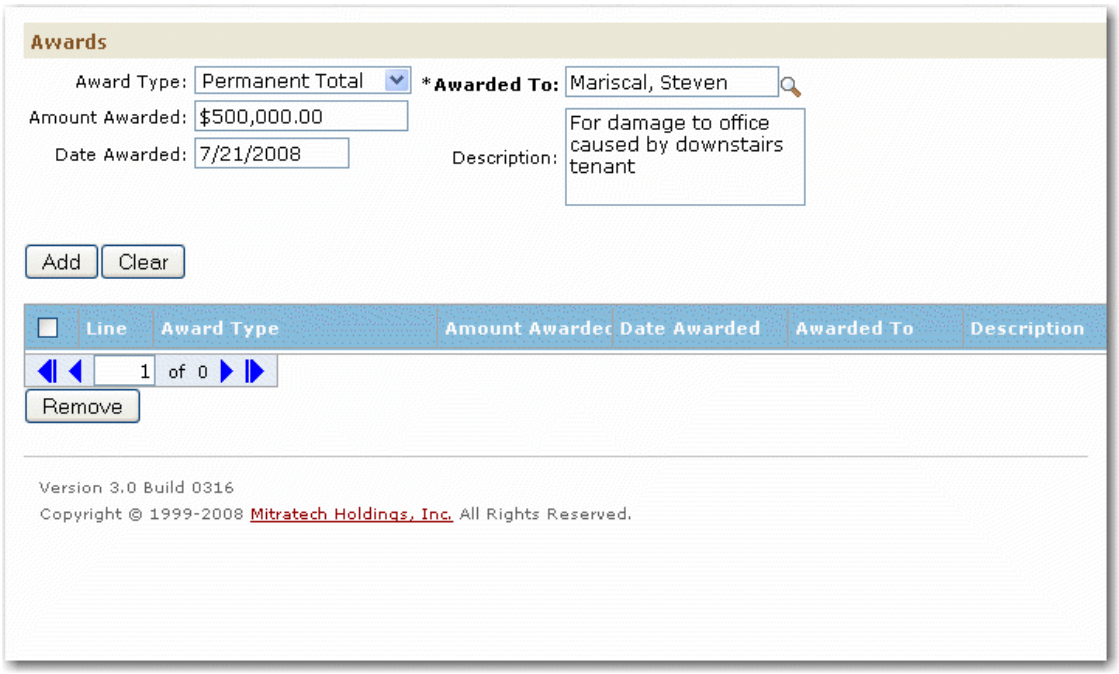
properties, you select a check box to designate it as contact-centric. Then, you specify the role of the related Contact. For details, see General Custom Object Information.

Note: *Only custom objects (including embedded custom objects) can be contact-centric.*

Results in End-User Interface

When you create a contact-centric custom object (for example, Loan), the relationship appears in the following areas of the end-user interface:

- **General block of object records**—Displays a contact search module field labeled according to the label you specify when you define the object as contact-centric (in this example, Applicant). End-users are required to select the object's main contact in this field.



Example: Contact-Centric Record (Edit Mode)

After the record is saved (and no longer in edit mode), the contact name appears as a link, for example, Mariscal, Steven.

- **Involvement tab**—Displays a hyperlink to the object record after the end user selects the contact in the object record.

Results in Designer User Interface

Whenever you create a contact-centric custom object definition (for example, Loan), the relationship appears in the following areas of the Designer interface:

- **Object Navigator window**--The contact-> attribute appears in all of the object definition's TProject tables, and you can use it to build attribute paths. For more details, see Using Object Navigator.
- **Unique ID tab**—Displays the contact attribute in the list of attributes (if you are adding attributes to a naming pattern). For details, see Adding Object Attributes.

1.1.3.5 Sub-Objects

There are certain items that contain information that pertains only to a certain record, and therefore exist only within that record. Because they are "owned" by a record, they cannot be accessed outside of it. In TeamConnect, these items are referred to as sub-objects. Categories, task and project assignees, contact addresses, emails, and skills are all examples of sub-objects.

Both system and custom objects can have sub-objects, and different objects have different sets of sub-objects. For example, custom objects have the sub-objects Categories, Assignees, and Relations. The sub-objects of the system object Appointment are Categories, Attendees, and Resources. The following table provides a complete list of the sub-objects that are available for TeamConnect objects.

Sub-object List

Object	Sub-objects			
	Categories	Assignees	Relations	Other
Account	x			
Appointment	x			Attendee Resource
Contact	x		x	Address Email Fax Internet Address Phone Rate Skill Territory
Document	x			
Expense	x			
History	x			
Invoice	x			Line Item
Task	x	x		

Custom Object	**x**	**x**	**x**	
Involved	**x**		**x**	
Embedded Custom Object	**x**			

Each sub-object has its own value set, or group of values, that must be defined in order to add that sub-object to a record. For example, to add a project assignee, the following values must be specified: an assignee (or user), a role, status (active or inactive), and the date of assignment.

Some values, such as status and date of assignment, are set automatically by the system. Others, such as assignee and role, must be entered by the end-user. In the end-user interface, sub-objects are added to a system or custom object record through a batch screen, where the data entry fields in each row indicate the values that are necessary for the sub-object. For details on using batch screens, see Working with Batch Entries.

If your design requires templates and wizards, it is important to know a sub-object's value set and which values have to be set manually in the template. If any required manual values are not included in a template, the sub-object will not be added by the wizard. For more details, see Adding Sub-objects to Templates.

1.1.3.5.1 Embedded Custom Objects

An *embedded object* is a simplified custom object created within (or embedded in) another custom object. They are similar to child custom objects, but have fewer capabilities.

A child object is a custom object in a hierarchical relationship with another custom object. It has all the functionality of a custom object (such as wizards, rules, assignees, and phases), and has a dynamic life cycle and workflow.

An embedded object has a hierarchical relationship with a custom object, but does not have workflow or a dynamic life cycle. Embedded object records are best suited for static information—information that is not likely to change as the parent record moves from phase to phase.

Embedded objects do not have a workflow or dynamic life cycle. As outlined in the Properties of Embedded and Child Object table, you cannot define phases, assignees, or rules in embedded objects. They are best suited for information that is not likely to change as the parent record moves from phase to phase. Because the information is owned by the parent record, it is not relevant to other records.

The following table compares embedded objects and child objects in key functional areas.

Properties of Embedded and Child Objects

Functional area or property	Embedded objects	Child objects
Accessible through **All Services**	No	Yes
Accessible through parent record	Yes	Yes

Can be contact-centric	Yes	Yes
Can have child objects	No	Yes
Unique ID	Yes	Yes
Auto naming patterns	Yes	Yes
Phases and phase transitions	No	Yes
Assignees	No	Yes
Categories	Yes	Yes
Custom fields	Yes	Yes
Blocks	No	Yes
Object views	No	Yes
Search views	Yes	Yes
Rules	No	Yes
Templates	No	Yes
Wizards	No	Yes
Related objects	No	Yes
Involved parties	No	Yes

1.1.4 Creating and Defining Objects

Creating and defining objects is the core procedure of customization and developing a business solution in TeamConnect. The properties you define for each object are consistently applied to all of the object's records, which are then seen in page form by end users.

For example, you can arrange data fields to be displayed with the same layout in each record. You can also configure an object to automatically name records with a pattern of data. This pattern might include the date on which the record was created, the record's category, and the name of user who created it.

Alternatively, you might also configure an object to automatically number records with a pattern of sequential numbers, the date the record was created, and even the record name. As you build it, step by step, each object displays, requests, and arranges the information you want for the end user, in the way you want it.

Defining Objects

Defining objects is a multi-step procedure. Depending on your design, some steps may not be necessary. However, no matter how many steps you use to implement your design, the task of defining an object always requires:

- Thorough understanding of your organization's business requirements and processes
- Clear vision of the final goal you want to achieve

  If an entity-relation diagram was created during the solution design stage, it may be very useful when defining objects. For more information, see Related Objects.
- Knowledge of TeamConnect terminology

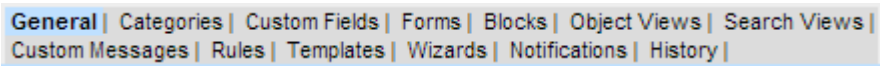
  As your solution is developed, you need to understand basic terms used in this documentation. Terms are introduced throughout this document, and detailed descriptions can be found in the Glossary.

1.1.4.1 About Object Definitions

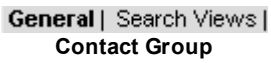
An *object definition* is a specification of all the properties of an object. When fully defined, an object definition contains all of the details about an object, including its name, the icon that represents it in record screens, its categories, its display, and the custom fields that contain or request information specific to your organization.

Object definitions determine the appearance of records and search screens, how wizards are used to create records, naming and numbering patterns for (custom object) records, and business rules triggered when users are working with object records.

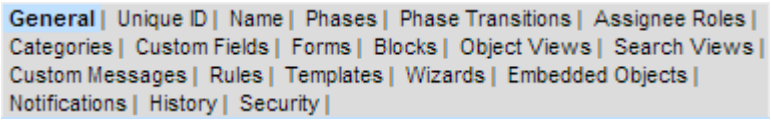
Each type of object definition screen has a specific set of tabs that allow you to manage different properties of the object:



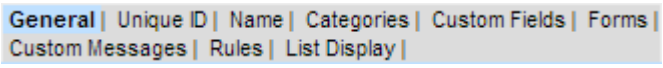
System Object Definition Tabs



Contact Group (Address Book), Group, and User System Object Definition Tabs



Custom Object Definition Tabs



Embedded Object Definition Tabs

Keep in mind that the end-user interface pages display:

- Record information and fields in the main or right pane
- Objects associated with the current record as links and collections (search views) in the left pane

Note: *Designer refers to Contact Groups, but the end-user interface calls the same objects Address Books.*

The following table provides a summary of the properties that can be defined for different objects through these tabs. Tabs are listed in alphabetical order and may appear in a different order in the interface.

System and Custom Object Definition Screen Tabs

Tab	Object type	Description
Assignee Roles	Custom	Allows you to create new assignee roles or view the existing assignee roles created for the selected object. See Defining Assignee Roles for Custom Objects.
Blocks	System and Custom	Allows you to create new custom blocks or view a list of all existing custom blocks defined for the object. See Blocks. (Does not apply to embedded object definitions.)
Categories	System and Custom	Allows you to define new categories or view the existing categories created for the selected object. See Using Categories.
Conditions	System and Custom	Allows you to create an object that you can use in routes to base the stops on conditional expressions. See Conditional Expressions.
Custom Fields	System and Custom	Allows you to create new custom fields or view the existing organization-specific custom fields created for the object. See Custom Fields.
Document Types	System (Document only)	Allows you to specify which document file types can be uploaded to TeamConnect.
Embedded Objects	Custom	Allows you to add new embedded objects or view a list of all existing embedded objects created for the selected parent object. See Embedded Custom Objects.

General	System and Custom	Allows you to specify general object information for custom object definitions. This information includes the object name and icon file. For details, see: • Creating Custom Objects • General System Object Information • Creating Custom Objects • Creating Embedded Objects
History	System and Custom	Allows you to access a search screen where you can search for the history entries associated with the selected object definition.
List Display	Custom (Embedded only)	Offers options for displaying embedded records in the parent object records.
Name	Custom	Displays two options for naming for records of the selected custom object based on a pattern of attributes. See Naming Patterns for Custom Object Records.
Non-US Tax Categories	System	Define non-US Tax Categories, including Codes and Descriptions.
Notifications	System and Custom	Allows you to customize which notification templates are sent at the object level. These settings will override the System Notifications settings in the TeamConnect Admin Settings. See Customizing Notifications for Object Definitions.
Object Views	System and Custom	Allows you to create new object views or view a list of all existing object views defined for the object to meet your organization's specific needs. See Object Views. (Does not apply to embedded object definitions.)
Phase Transitions	Custom	Allows you to create new transitions or view the existing transitions between the phases created on the **Phases** tab, that is the sequence in which the project phases follow each other. See Defining Phase Transitions for Custom Objects.
Phases	Custom	Allows you to define new phases or view the existing phases created for the project life cycle of the selected custom object. See Creating Phases in Custom Objects.

Rules	System and Custom	Allows you to create new rules or view the existing rules defined for the selected object. See Using Rules.
Search Views	System and Custom	Allows you to create new search views or view a list of all search views created for the object. See Creating Search Views.
Security	Custom	Allows you to set how record-level security and restrictions of the custom objects' records should apply to their related records.
Templates	System and Custom	Allows you to create new templates or view a list of all existing templates created for the object. See Using Templates.
Unique ID	Custom	Displays several options for setting a unique identifier for records of the selected custom object, for example, auto numbering patterns. See Unique Identifiers for Custom Object Records.
Wizards	System and Custom	Allows you to create new wizards or view a list of all existing wizards created for the object. See the Creating Wizards.

1.1.4.2 Viewing Object Definitions

You may access system object definitions and existing custom object definitions in the **Object Definition List** screen. The **Object Definition List** screen (Object Definition List image) provides a snapshot of the existing object definitions. This screen summarizes their general information and relationships to each other.

You may sort the object definition list into ascending or descending order by clicking the **Object Definition** label.

To open an existing object definition

1. From the **Go to** drop-down list click **Object Definitions**.

   The **Object Definition List** screen appears.

Object Definition List

2. Click the hyperlink of the object definition you want to open.

   The **General** tab of the selected object definition appears.

   Click the tab where you want to view, add, or modify the object information. For details, see About Object Definitions.

Although Involved as well as embedded object definitions appear in the list, they can only be created from within their parent custom object definition. For more details on creating custom objects, see Creating Custom Objects.

The following table describes the items in the object definition list.

Object Definition List Screen

User interface element	Description
New	Displays a new custom object definition screen with empty fields where you can define a new custom object. For more details on the definition procedure, see Creating Custom Objects.

	Note: *You cannot use this button to create system objects, nor embedded custom objects. Embedded custom objects, Milestone and Involved objects can be created only from within their parent object definitions.*
Refresh	Click to refresh the list after adding object definitions or deleting existing ones.
Object Definition	• Displays a list of all available object definitions, including MilestoneInvolved, and embedded object definitions, in a hierarchical tree structure. The display structure and functionality is similar to that of the Windows® Explorer® where you can collapse and expand parent nodes that have sub-nodes within them. *Note: The hierarchical relationships are reflected only for custom object definitions.* • Click hyperlinks to access object definition screens where you can view and modify the necessary properties through the respective tabs. For more details on the tabs and associated properties, see the System and Custom Object Definition Screen Tabs table. • The types of the displayed object definitions are indicated by the following icons: —System object definition —Custom object definition —Embedded custom object definition • Click the column heading to sort the list alphabetically, according to the first, top-level nodes in the tree structure.
Unique Code	Displays the unique codes of the object definitions. Click the column heading to sort the list alphabetically.
Contact-Centric	Indicates which custom object definitions are contact-centric: • If a custom object definition is contact-centric, the role of the main contact appears. • If not, the column remains empty. This column is not sortable.

1.1.4.3 Configuring System Objects

By default, some properties in system objects are already defined. For instance, most system objects are named and have unique codes. For the most part, defining these objects is a matter of configuring general system object information and defining categories, search views, and rules. For details, see Customization Sequence and General System Object Information.

If your design includes invoices, you may need to configure line item views according to the organization's requirements. For details, see Customizing Invoice Line Item Views.

Defining Contact Group, User Account, and Group Account Objects

There are three TeamConnect system objects, which unlike other system objects, have a simplified object definition procedure. They are Contact Group (called Address Book in the end-user interface), User, and Group objects. Because of their functionality, they do not require categories, custom fields, rules, or wizards. This functionality is reflected in the set of tabs they have in their object definition screens.

Contact Group (Address Book), Group, and User System Object Definition Tabs

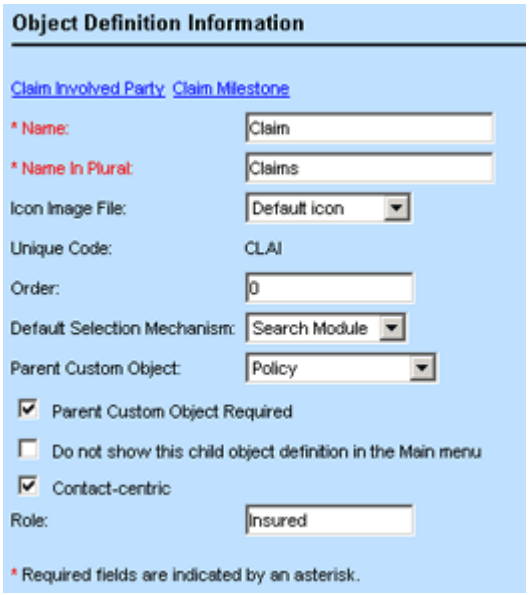
To define the simplified object definition objects

1. Select an **Icon Image** file on the **General** tab of the selected object. See the General Tab on Custom Object Definition Screens table.
2. Customize the default **Search Views** provided with the TeamConnect application or create your own. For details, see Creating Search Views.

The Contact Group (Address Book), User, or Group object is defined.

1.1.4.3.1 General System Object Information

For system objects, the **General** tab of the object definition screen includes identification information, such as object name in the singular and plural forms, unique code, and an icon image file. Depending on the system object, some of these details may not be modified.



General Tab on Custom Object Definition Screens

See the **General** Tab on **Custom Object Definition Screens** table for details about the **General** tab of object definition screens. In addition to the General page fields in the table, the **Expense**, **Invoice**, and **Task** system objects include the following check-box:

Allow edits to custom fields after posting—Place a checkmark in this check-box for the **Expense**, **Invoice**, or **Task** object to allow users to edit custom fields after posting a record for that object.

See Customization Sequence for customizing a system object.

1.1.4.3.2 Default Object Icons

By default, each record in the TeamConnect end-user interface appears with a certain icon that identifies the object to which it belongs. The images of these default icons are automatically generated from the respective Java class files and are assigned to each object. The following are the default images that are used for each object:

Account	**Contact**	**Invoice/Line Item**	**User**	**Involved**
Appointment	**Document**	**Expense**	**User Group**	**Milestone**
Approval	**History** **No default icon**	**Task**	**Project** **(Custom Object)**	**Contact Group (Address Book)**

			No default icon	

These icons appear at the top of the record window, for example:

If you want to use your own images, upload the appropriate image files to the **Icons** folder located in the **Top Level** directory in the Documents area.

1.1.4.4 Creating Custom Objects

The process of defining custom objects starts with creating a custom object, defining its general properties, and saving the object. Note that this process may be a complex and lengthy procedure.

Note: *Before users can view custom objects, you must assign the corresponding object rights to their group. Each Custom Object has its own set of rights.*

There are two approaches to creating a custom object: Copy an existing object definition, or create an object definition from scratch.

If the object to be created will have a definition similar to an existing object definition, you can copy the definition of the existing object, then modify the copy to reflect the exact properties you want in the new object definition. This may save time compared to creating the entire new object definition from scratch.

To create a custom object by copying an existing definition

1. In the Designer, go to **Object Definitions**.
2. Click on the existing object definition name.
3. In the resulting page, click **Create a Copy**.
4. A new page appears, showing general information for the new definition. You should immediately change the following field values:
   - Name
   - Name in Plural
   - Unique Code (the present value is the same as the one from the existing definition, which would cause an error if not changed)
5. Enter other appropriate general information in the fields as described in the General Tab on Custom Object Definition Screens table.
6. **Save** the object definition.

To create a custom object from scratch

1. To create a new custom object definition, select one of the following options:

- Click **New** in the **Object Definition List** screen.
- In the Designer window, click **Create a new**, and then select **Object Definition**.
- A new object definition screen appears with a blank **General** tab displayed by default.

2. Enter the appropriate general information in the fields as described in the General Tab on Custom Object Definition Screens table.
3. Save the object definition.

Continue defining object properties

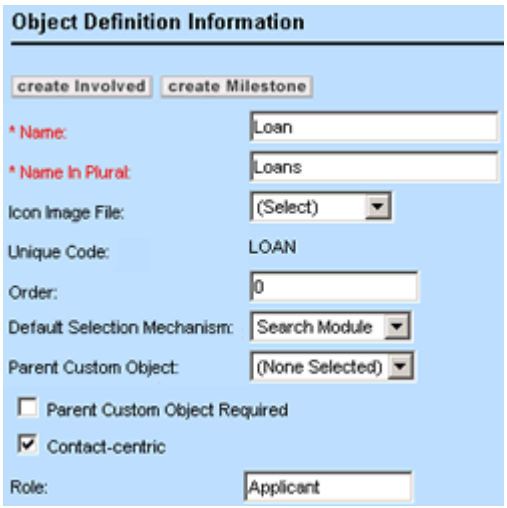
Regardless of which approach you used to create the custom object, you should now continue defining its properties as explained in Customization Sequence.

Once the object properties match your solution design, the object is defined and displayed on the **Object Definition List** screen. For details, see Viewing Object Definitions.

Resulting User Interface and Other Changes

When you save the custom object information, the following events take place:

- You see the **Create Involved** and **Create Milestone** buttons at the top of the **General** tab.



Create Involved and Create Milestone Buttons

- Two document folders with the object's name are automatically created in the Documents area and added to the following directories:
  - **Top Level/System/Object Definitions/*Object Name***

    This folder stores all documents related to the design of the object definition, such as XML files for custom blocks, Involved and Milestone object definition files, document templates, and rules.
  - **Top Level/Attachments/*Object Name***

    This folder stores all of the files that are uploaded to the object records.

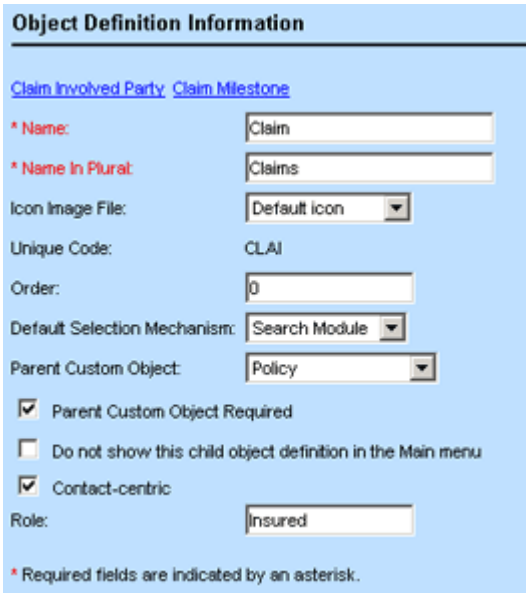
Important: *For each object record, this folder is created only when the record is saved and end users access the* ***Documents*** *block of that record for the first time.*

For more details on document folders, see the *Administration Guide*.

For a complete overall procedure of defining custom objects, see Creating Custom Objects.

1.1.4.4.1 General Custom Object Information

The **General** tab of a custom object definition screen displays general information about the object as described in the General Tab on Custom Object Definition Screens table.



General Tab on Custom Object Definition Screens

To add and save a custom object you must define its name in the singular and plural forms and specify its unique code. You may also specify the rest of the general information according to your design. For instructions on how to define a custom object, see Customization Sequence.

The following table describes the items on the **General** tab of custom object definitions.

General Tab on Custom Object Definition Screens

Item	Description
customObject Involved Party **Milestone**	These hyperlinks provide access to the custom object's Involved and Milestone object definitions. They are only displayed once you define these objects. When you add a custom object, you cannot create Involved or Milestone objects for it until you save the custom object definition. When you save the custom object definition for the first time, you see the **Create Involved** and **Create Milestone** buttons (Create Involved and Create Milestone Buttons image).

Name	Type a name for the custom object. The maximum length is 50 alphanumeric characters. ***Important:*** *The name must not start with a number and cannot contain special characters.* This name appears in the end-user interface and names the corresponding Document area folder created for the object definition (see Resulting User Interface and Other Changes). Any special characters entered in this field are automatically converted to hyphens in Document area folder names. For example, **Matter:Issue** appears as **Matter-Issue**.
Name in Plural	Type the plural form of the custom object name. The maximum length is 50 alphanumeric characters. The plural name appears in the end-user interface where appropriate.
Icon Image File	Select the image file to appear in the following locations when users access records of this object: • Top of the record screen, next to the record name • On the object's task bar buttons • In the object's search screens You must upload all custom image files to the System>Icons folder at the **Top Level** level in the **Documents** area. Otherwise, the default image is used. All default images are generated from the class files and are not located in the **Icons** folder. For more details, see Default Object Icons.
Unique Code	Enter a four-character alphanumeric combination to uniquely identify this custom object. You can type the unique code in upper or lowercase. However, it is saved and displayed in uppercase. When you save the custom object, the unique code is always displayed as read-only and you cannot change it. The code must be unique for all objects, including Involved and Milestone objects, which are created from within custom objects. The unique code is used as the tree position for the root category of the custom object in the custom block XML files, XML layer, API rules, and Document Generator.
Order	Type an integer to specify where you want the name of this custom object to appear in ordered lists in the end-user interface.

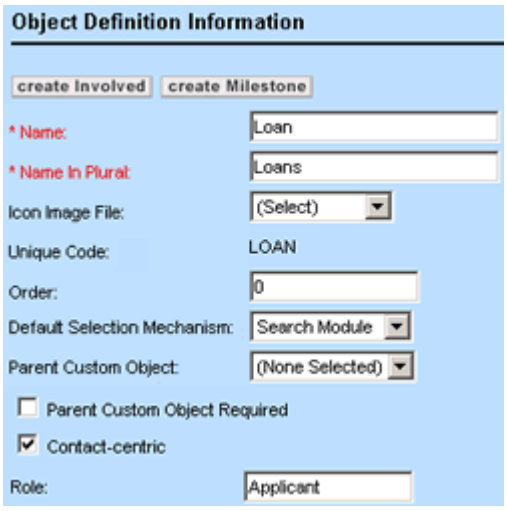
	Each level in the parent-child hierarchies is sorted separately according to the order specified. The objects with the same display order are sorted alphabetically.
Default Selection Mechanism	Specify how the end user will select the object's records when they are referenced in other object records. For example, in the **Parent** field on the **General** tab of child projects, or in custom fields of type Custom Object. The available options include: • **Drop-down list—**Recommended for listing a small number of records. • **Search Module—**Recommended for a large number of records, as it may take a long time to load the list.
Parent Custom Object	Select a custom object that must be set as a parent for the object you are defining. ***Important:*** *This is not a mandatory selection, as it depends on your business model.* For details, see Parent-Child Relations between Custom Objects.
Parent Custom Object Required	Select this check-box if having a parent custom object is mandatory for the custom object you are creating. See also Parent-Child Relations between Custom Objects.
Do not show this child object definition in the Main menu	This field is available only for custom object definitions that require a parent. Select this check-box if you do not want users to create and access the records of this object from the drop-down lists in the end-user interface, or to avoid unnecessary clutter in the menus. This option does not affect user group rights to create or read these records. They can still do so from the respective parent records.
Do not show New button in related object block	Select this check-box if you want to hide the **New** button that provides a drop-down list of wizards in the related object block. For more information about the drop-down lists of wizards, see Testing and Troubleshooting Wizards.
Contact-centric	Select this check-box to make the object contact-centric. When you select this check-box, the Role field appears.

	Note: *This setting is optional and should be based on your organization's needs.* For details, see Contact-Centric Custom Objects.
Role	Enter the role of the contact on which the custom object is dependent. This becomes a field label on the General tab of all object records in the end-user interface. For details see Contact-Centric Custom Objects. This field appears only if the Contact-centric check-box is selected.
Single Object Instance Only	Select this check-box to specify that the custom object can have only one record in the end-user interface. When the user clicks the link for the custom object, TeamConnect automatically opens the only record. If the record does not exist, the user must create a record first.

1.1.4.4.2 Creating Custom Objects

The Milestone and Involved custom objects are system objects that can be created and accessed only from within Custom Objects in Designer. For more information, see Unique Custom Object Types.

Whenever you add a new custom object, you see the **Create Involved** and **Create Milestone** buttons at the top of the **General** tab.

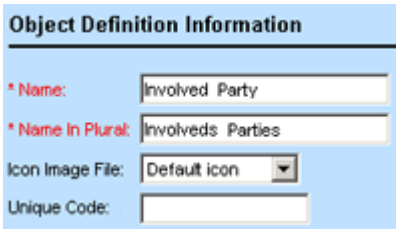


Create Involved and Create Milestone Buttons

To create a Milestone or Involved object

1. Open the **General** tab of the custom object definition where you want to create the appropriate object.
2. Click either the **Create Involved** or **Create Milestone** button.

   The corresponding object definition screen of the selected object appears with the **General** tab displayed by default. The **Name** and **Name in Plural** fields are automatically populated:



General Tab on Involved Party Object Definition Screen

3. (Optional) Type the desired name in the singular and plural for the object.
4. (Optional) From the **Icon Image File** drop-down list, select the image file that appear when the user accesses the object records. Otherwise, the default image is used instead.

   All image files are located in the **Icons** folder at the **Top Level** level in the **Documents** area.
5. In the **Unique Code** field, enter a four-character alphanumeric combination to uniquely identify this object.

   This code is unique for each of these objects, regardless of the custom object it belongs to.
6. Save the object definition and continue defining its properties as described in Configuring System Objects.
7. When finished defining the object's properties, save the definition and click the **Refresh** button at the top of the **Object Definition** page.

   The Milestone or Involved object is defined.

Resulting user interface and Other Changes

When you save Involved or Milestone object information, the following actions take place for each object:

- The object's name appears as a hyperlink at the top of the **General** tab of the respective custom object.
- Two document folders with the object's name are automatically created in the Documents area and added to the following directories:
  - **Root/System/Object Definitions/*customObjectName*/*milestoneName***

    or

    Top Level/System/Object Definitions/*customObjectName*/*involvedName*

This is the folder where all documents related to the design of the object definition are stored, for example, XML files for custom blocks, appropriate object definition files, document templates, and rules.

- **Root/Attachment/*customObjectName*/*recordUniqueID*/MilestoneList**

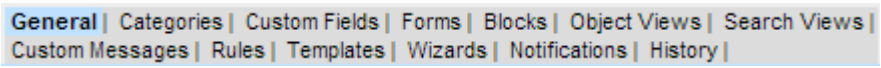
  or

  Top Level/Attachment/*customObjectName*/*recordUniqueID*/InvolvedList

  This is the folder where all files uploaded to the object records are stored.

  Important: *For each Involved and Milestone record, this folder is created only when the record is saved and end users access the* ***Documents*** *block of that record for the first time.*

Milestone and Involved objects have the same set of object definition tabs as system objects and their object definitions can be customized like those of system objects. See Configuring System Objects.



System Object Definition Tabs

The roles of the involved parties are defined through the **Categories** tab of the Involved object definition screen. See Using Categories.

Group rights to the Milestone and Involved objects are identical to those of their parent object. For example, if a group lacks access rights to the parent object, it cannot access that object's Milestone and Involved records. Likewise, search results for such records are not displayed.

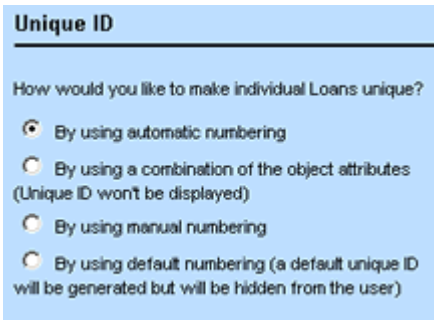
1.1.4.4.3 Unique Identifiers for Custom Object Records

Unique identifiers are unique combinations of alphanumeric characters assigned to each custom object record when it is created. Users can view unique IDs in the end-user interface if the identifier is set to be displayed.

You can assign unique identifiers for custom object records:

- By using auto numbering. See Creating Auto Numbering Patterns.
- By using object attributes. See Creating Unique Identifier Patterns from Object Attributes.
- By using default numbering

You can select the desired options for the project identifiers through the **Unique ID** tab of the respective custom object definition.



Unique Identifier Options

The following table explains each available option in detail.

Unique Identifier Options

Option	Description
By using automatic numbering	Select this option to have TeamConnect automatically generate numbers for object records according to the specified pattern. When a new record is being created, the **Number** field will have the word **Auto** displayed in parentheses. When the record is saved, the **Number** field displays the assigned sequential number. See Creating Auto Numbering Patterns.
By using a combination of the object attributes	Select this option to create a pattern consisting of the object attributes that, taken together, will uniquely identify the individual object records of the selected custom object. No numbering will be displayed in the end user interface. See Creating Unique Identifier Patterns from Object Attributes.
By using manual numbering	Select this option to allow users to enter their record numbers manually. When a new record is being created, the **Number** field will be displayed as an empty text field where the user can enter the desired number. ***Tip:*** *Remember to select the desired display format for the project hyperlink. See Project Link Display Format.*
By using default numbering	Select this option to have primary keys automatically assigned to each project and stored in the database. No numbering will be displayed in the end-user interface.

Depending on the option you choose at the top of the **Unique ID** tab, the bottom part of this screen may display an additional section where you can specify the desired properties.

1.1.4.4.3.1 Creating Auto Numbering Patterns

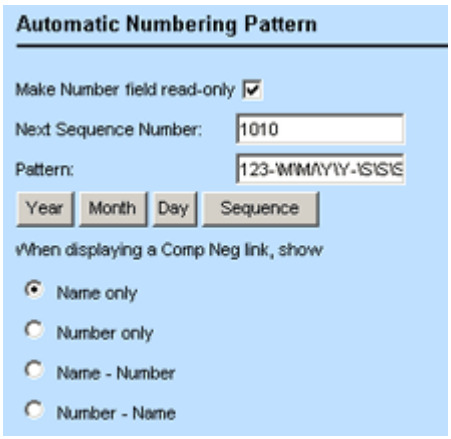
You can specify an automatic numbering pattern for generating individual record IDs for custom objects. A pattern can consist of a combination of the following elements:

- The date when the project is created
- A sequential number, which differentiates a particular project from the other projects created on the same day
- Any static information, such as a department name or special characters used as format separators

To set an auto numbering pattern for projects

1. Open the **Unique ID** tab of the appropriate custom object definition.
2. Select the **By using automatic numbering** option.

   The **Auto Numbering Pattern** section appears at the bottom of the screen.



Auto Numbering Pattern Settings

3. Fill in the appropriate fields as described in the Auto Numbering Pattern Settings table.
4. Select the format for displaying the project hyperlink.
5. Click **Save**.

   TeamConnect uses the new auto numbering pattern to assign a number to each new custom object record.

If you change the auto numbering pattern of an object definition, any existing records keep the previous unique identifier conventions until users update the records, at which point the new auto numbering pattern is used.

Updating the auto numbering pattern does not count as updating the object definition, so record locking is not enforced. However, updating other fields of the object definition causes an optimistic locking error.

The following table describes the auto numbering pattern settings.

Auto Numbering Pattern Settings

Field or control	Description
Make Number field read-only	Select this check-box to make the project number uneditable. ***Important:*** *If the number is editable, the user will have the ability to modify the number. These manual changes will take precedence over the automatic pattern.*
Next Sequence Number	Enter a starting number for the sequential part of the auto numbering pattern. By default it is set to 0 or, if auto numbering has already been enabled, the next number in the sequence.
Pattern	Enter a pattern for the number, using any of the following elements: • Alphanumeric characters • Special characters, such as $, &, *, ~. • Spaces • Dates • Sequence characters (see the buttons below)
Year	With the cursor in the **Pattern** field, click this button to add a two- or four-digit year to the **Pattern**, for example, \Y\Y.
Month	With the cursor in the **Pattern** field, click this button to add a two-digit month to the **Pattern**, for example, \M\M.
Day	With the cursor in the **Pattern** field, click this button to add a two-digit day to the **Pattern**, for example, \D\D.
Sequence	With the cursor in the **Pattern** field, click this button to add a multi-digit sequential number to the **Pattern**, for example, \S\S\S\S\S\S\S. It is a good idea to base the number of digits that you include in the sequence on the number of records that are created in a day. For example, if your organization creates 100 claims per day, you might want to consider making the sequence consisting of three digits (\S\S\S). ***Important:*** *You must include the sequence part into the* ***Pattern****, to make sure that the project number is unique.*
When displaying a	Select the appropriate choice button to specify which format to use when displaying record hyperlinks of the object in other object records. See Project Link Display Format.

CustomObject link, show	

1.1.4.4.3.2 Project Link Display Format

Whenever a custom object record is referred to in other records or appears, for example, in search screens, it appears as a hyperlink. This hyperlink appears in one of the four available formats:

Project Link Display Options

Option	Example
Name Only (Default)	Johnson v. Matthews
Number Only	2001-1018-003
Name - Number	Johnson vs. Matthews-2001-1018-003
Number - Name	2001-1018-Johnson vs. Matthews

You may select the desired format on the **Unique ID** tab of the custom object for the automatic and manual numbering options. Existing records keep the previous settings only until a user makes changes or updates then. When a user updates an existing record, the new setting you specified takes effect.

1.1.4.4.3.3 Creating Unique Identifier Patterns from Object Attributes

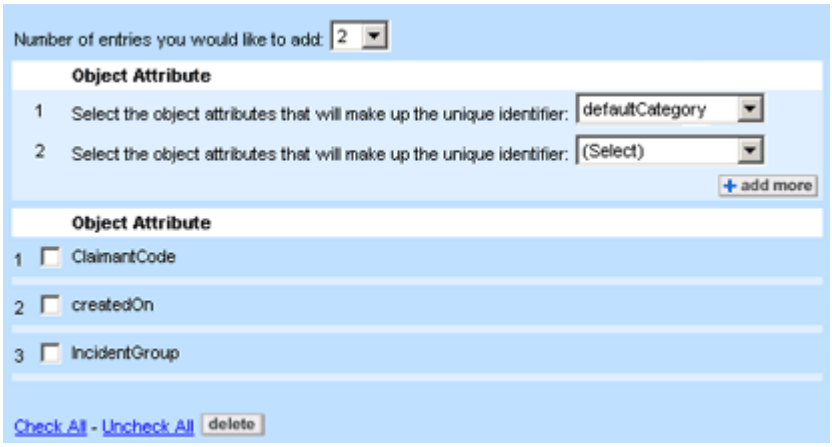
You can specify a sequence of custom object attributes to uniquely identify individual custom object records. This can be useful to ensure that no duplicate records are created. For example, if you want to prevent duplicate matter records from being created for the same person involved in a dispute, you can create a unique identifier pattern that consists of the person's name, the type of dispute, and an incident date. In this way, you ensure that no two matter records with the same details are created for the same dispute.

For more information on attributes, see Using Object Navigator and Object Model: Read This First plus the additional reference tables this reference points you to.

To set a unique identifier pattern for projects

1. Open the **Unique ID** tab of the appropriate custom object definition.
2. Select the By using a combination of the object attributes option.

   The **Object Attributes** section appears.



Object Attributes in Unique Identifiers

3. Select the **Number of entries you would like to add** from the drop-down list.
4. From each drop-down list, select the desired object attribute.
5. If you want to add more attributes, click **add more**. Otherwise, click **Save**.

   The object attribute pattern you defined for the custom object takes effect immediately. Existing records keep the previous unique identifier conventions until a user updates them.

Points To Remember

Keep the following points in mind when creating unique identifier patterns from object attributes:

- Object attribute patterns are stored in the database but they are not displayed in the end user interface.
- All object attributes you select for a pattern are added alphabetically one after another, no formatting is used.
- The attributes that you can select for the unique identifier patterns are automatically preselected by the system for you. They often include a few system fields, such as **contact** (applicable for contact-centric object definitions), **name**, **nameUpper**, **defaultCategory**, **numberString**, **numberStringUpper**, and all custom fields under the Root category.
- Even though taken in isolation certain attributes may never constitute a unique pattern (for example, **name**), taken together, they do.
- All object attributes you select for a pattern automatically become required fields for the end user.
- If you add custom fields to the pattern, remember to always include those fields in the custom pages you create.

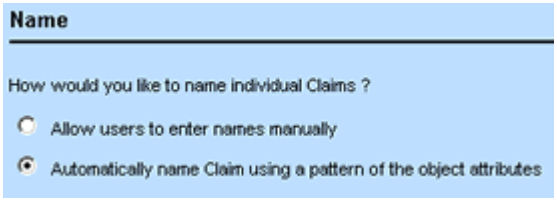
1.1.4.4.4 Naming Patterns for Custom Object Records

Unlike unique identifiers, such as record numbers and object attribute patterns, that serve to uniquely identify each record in the database, names in custom object records are intended to provide a description of each record that may or may not be unique.

Depending on your organization's naming conventions, you can assign names to custom object records manually or automatically. Using an automatic naming pattern for project record names is useful when:

- Your organization has project naming conventions. For example, each dispute record must be named after the contact of the person or company that your organization is representing, the contact of the opposing party, and the type of dispute.
- Naming patterns can be identified in your organization's project names, even if there are no specific naming requirements or conventions.
- Spelling, or other errors and inconsistencies in record names may severely affect the business process.

You can set the record naming options for a custom object through the **Name** tab of its object definition.



Project Naming Options on Name Tab

The following table describes the naming options.

Project Naming Options

Option	Description
Allow users to enter names manually	Select this option to have the user manually enter names for project records. When a new object record is being created, an editable **Name** field will be displayed for the user.
Automatically name customObject using a pattern of the object attributes	Select this option to define a naming pattern that will be automatically assigned to all custom object records. The name is automatically displayed as a read-only field after the project is saved.

With the auto naming feature for custom object records, you can construct the desired pattern according to the standards and conventions accepted in your organization.

Example

Consider an automatic naming pattern for a custom object called Loan where each record must be named after the contact who applied for the loan, the type of loan applied for, and the amount of money requested. The format for displaying this information would be: Applicant's LoanType - LoanAmount.

To create a pattern like this, do the following actions:

- The name of the applicant from one of the following elements:
  - A custom field of type Involved that contains the applicant's name
  - The contact attribute that contains the main contact's (applicant's) name, which is available only in contact-centric custom objects

  By default, all persons' names appear in the following format: Last Name, First Name.
- The default category of the custom object that contains the loan type
- The custom field that contains the loan amount information
- The following **literals**:
  - Apostrophe "s"
  - Hyphen
  - Three spaces between the attributes and literals

  Tip: *Whenever you have spaces around literals, add these spaces together with their literals, for example, a space after the 's, and two spaces before and after the hyphen. They are added even if you do not see them in the item list.*

When you add the information to the name pattern of the custom object, TeamConnect automatically creates the name for each record when the user saves it. This way, all records are named correctly and consistently with your organization standards, based on the attributes you select.

Points To Remember

Consider the following points when creating naming patterns for objects:

- The naming pattern of a record will reflect only those attributes that have values. If an attribute in a naming pattern has no value, it will not be displayed in the record name.

  Tip: *To ensure that all record names are properly displayed, you can make the attributes required. If you specify any custom fields, you must include them in your custom pages.*

- Naming patterns must contain at least one attribute. They cannot consist only of literals.
- Project names support a maximum of 250 characters. If the data stored in the selected attributes and literals exceed 250 characters, the name of the project is truncated.

  Tip: *If you have added long strings to the naming pattern, consider abbreviating them.*

- To have a person's name displayed in the naming pattern, specify a custom field of type Involved, or the contact attribute for contact-centric objects.
- By default, all person contacts' names will be displayed in the format Last Name, First Name. If you do not want to display the comma between the names or you want to change their order, you can use two attributes.

For example, instead of selecting the **contact** attribute for a contact-centric object, you have to add it twice and traverse further to select an attribute for each of the names:

- `contact.firstName`
- `contact.name`

You may define naming patterns by selecting the **Automatically name *custom object* using a pattern of the object attributes** option button on the **Name** tab of the custom object definition. For instructions, see Creating Auto Naming Patterns.

1.1.4.4.4.1 Adding Object Attributes

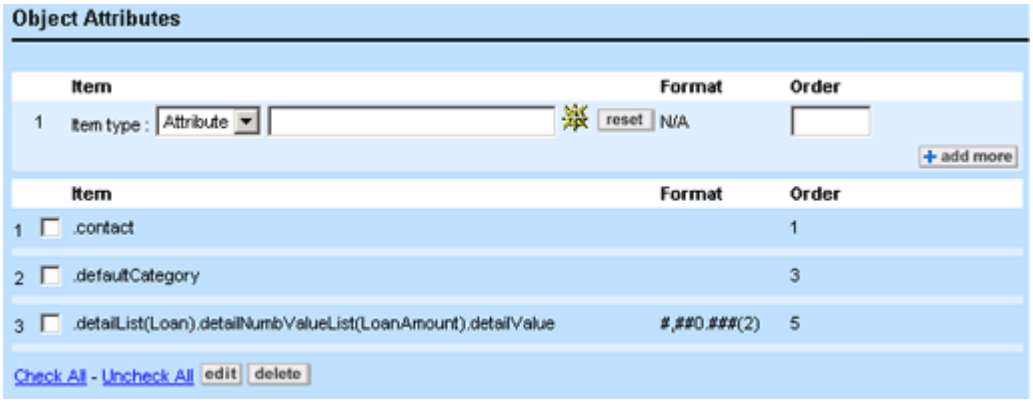
Attributes are the building blocks of objects and contain the actual data you enter into TeamConnect. For example, you may use attributes in naming patterns for custom object records to ensure that certain values are consistently used.

Naming patterns can include such values as:

- Name of an involved party
- Specific dates
- Categories
- Names of other objects, such as parent objects or other custom objects

You can select the attributes you want to use in your naming patterns from the **Object Attributes** section of the **Name** tab of the custom object definition. Before you begin defining naming patterns for custom objects, you must have an understanding of how to use Object Navigator.

For more information, see Using Object Navigator and Object Model: Read This First plus the additional reference tables this reference points you to.



Object Attributes on Name Tab

The following table describes the items in the **Object Attributes** section of the **Name** tab.

Object Attributes on Name Tab

Field	Description
Item type	Provides the following options: • Attribute • Literal

	• Character space Select **Attribute** to define attributes for the naming pattern.
Object Navigator	Click the **Object Navigator** icon. In the pop-up window, navigate to the desired attribute that contains the value you need for the pattern and click **ok**. The path to the selected attribute appears in the **Item** field. See Using Object Navigator.
Reset	Click to clear the contents of the **Item** field.
Format	Select the appropriate format for the selected attribute. For different attribute types, different **Format** data entry fields appear. For more information, see Attribute Format Fields.
Order	Enter an integer to indicate the **Order** of the attributes selected. The integer reflects exactly where in the naming pattern you want the attribute to be displayed (beginning, end, or middle of the name). Items with the same display order are sorted and displayed alphabetically according to the first letter of the attribute selected from Object Navigator (for example, "c" in contact, not "S" in Smith).
Item List	Displays the object attributes (and literals) that are already selected to be used in the project name pattern. Select the check-box of the desired attribute and click either the **edit** or **delete** buttons.

1.1.4.4.4.2 Attribute Format Fields

Depending on the type of attribute you select using Object Navigator, you can apply various display formatting options. For example, if you want the value of an attribute of type String to be displayed in upper or lowercase in each object record name, you select the appropriate format.

You can specify the following attribute format types in naming patterns:

- Boolean
- Date
- Number
- String

The following table describes the formatting options. For each of the object attribute types, two **Format** data entry fields appear.

Note: *If you decide to select no formatting for the attributes, the default settings are used instead as indicated in the table.*

Format Fields for Object Attributes on Name Tab

Attribute type	Format option	Description
Boolean	**True value**	Enter true or false values in the appropriate data entry fields for **Boolean** values, which are most commonly displayed as check-boxes or choice buttons. For example, if "isRegistered" custom field is the selected **Boolean** attribute, you may want to enter "Yes" for the true value and "No" for the false value. **Default Value:** False.
	False value	
Date	**Pattern**	Select month, day, and year format for the date attribute. **Default Format:** The settings selected on the **Preferences** tab of the **System Settings** screen.
	Separator	Select the symbol you would like to act as a separator between the month, day, and year of the date attribute. **Default Format:** The settings selected on the **Preferences** tab of the **System Settings** screen.
Number	**Pattern**	Select the format for the number attribute. The display of the numbers is based on the locale. For example, in Europe the number 1.234,56 has a period (.) as a grouping symbol and a comma (,) as the decimal point. In the United States, the number 1,234.56 has a comma (,) as a grouping symbol and a period (.) as a decimal point. **Default Format:** The US standard, where the comma is a grouping symbol.
	Decimal Places	Select the number of digits you would like to appear after the decimal point of the number attribute. **Default Format:** The settings at the time when the attribute value was saved in the database.
String	**Case**	Select either **UPPERCASE** or **lowercase** to format the string attribute accordingly. **Default Format:** The settings at the time when the attribute value was saved in the database.

	Abbr	If you want to abbreviate a string, select the desired abbreviation option. Whichever option you select, a blank data entry field appears for you to enter an integer. Depending on the option, the integer you enter specifies the following information: • When **First X Characters** is selected—the integer specifies the first number of characters from the beginning of the string. For example, if the string attribute consists of a contact name, such as Smith, John, and integer is **3**, the string will be formatted as SMI. • When **Acronym** is selected—the integer you enter specifies the first number of characters from each word in the string. For example, if the string attribute consists of a contact name, such as Smith, John, and integer is **3**, the string will be formatted as SMIJOH. ***Note:*** *If an integer greater than the number of characters in the string is entered, all of the characters will be included in the abbreviation.*

1.1.4.4.4.3 Adding Literals

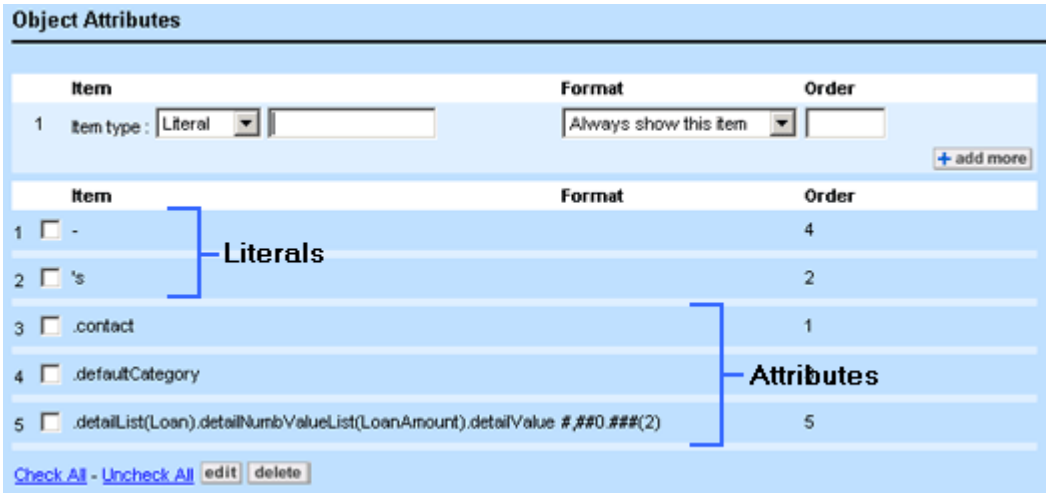
A literal is an alphanumeric or special character that is used as a separator between two attributes, or as a prefix or suffix to an attribute.

For example, if a user selected **Smith, John** as the applicant in a loan record, set **Loan** as the default category, and entered 20,000 as the loan amount, and the three attributes containing these values are added to the Loan naming pattern, the name of the project would be saved as Smith, JohnLoan20,000. Adding literals to the three attributes would create a clearer name for your users.

For example, you can add an apostrophe "s" with a space after it ('s) as a literal after the contact attribute and a hyphen with two spaces around it. After adding these literals, the name of the project would be saved as Smith, John's Loan - 20,000. Otherwise, you can add a space as a separate character.

Important: *A name cannot consist of literals only, it must have attributes.*

You can select the literals you want to use in your naming patterns from the **Object Attributes** section of the **Name** tab of the custom object definition.



Literals and Object Attributes on Name Tab

The following table describes the items in the **Object Attributes** section of the **Name** tab that you use to specify literals.

Literals and Object Attributes on Name Tab

Field	Description
Item type	Select **Literal** to add a literal to the record naming pattern.
Literal text field	Enter the appropriate characters to create either a separator between two attributes, or a prefix or suffix for an attribute. Tip: To add spaces between literals and attributes, type the space together with the literal, for example, a space before and after a slash. To add a space between two attributes, you must add it as a separate literal called **space character**.
Format	Select the format for the literal by selecting one of the following options: • **Always show this item—**The literal always displays as part of the name when the user saves the record. For example, if creating a literal "**record**," you may want the word "record" to always appear in the name. • **Don't show if the previous item is null—**The literal will not display if the object attribute before it has no value or is non-existent. For example, if creating a literal "**'s**," you may not want the "**'s**" to appear if there is no name in front of it.
Order	Enter an integer to indicate the display **Order** of the literals. The integer you enter reflects exactly where in the naming pattern you want the literal to be displayed (at the beginning or end of an attribute, or between two attributes). Items with the same display order are sorted and displayed alphabetically.

Item List	Displays the object attributes and literals that are used in name pattern for projects. Select the check-box of the desired attribute or literal and click either the **edit** or **delete** buttons to make changes.

1.1.4.4.4 Creating Auto Naming Patterns

To create an auto naming pattern for a custom object

1. Review the Points To Remember.
2. Open the **Name** tab of the appropriate custom object definition where you want to create a record naming pattern.
3. Select the **Automatically name *custom object* using a pattern of the object attributes** option.

   The Object Attributes section appears at the bottom of the screen.
4. Depending on your needs, do of the following actions:
   - Specify the object **Attributes** whose values you want to appear in the object records' names, as described in the Object Attributes on Name Tab table.
   - Specify the **Literals** that you want to add to the naming pattern of the custom object, as described in the Literals and object Attributes on Name Tab table.

   Caution: *If the data stored in the attributes and literals you select have a combined amount of over 250 characters, the name of the project is truncated, cutting off the information that appears after 250 characters.*
5. If you want to continue adding items, click **add more**. Otherwise, click **Save** on the toolbar.

   The record naming pattern you defined for the custom object takes effect immediately. Existing records keep the previous naming conventions only until a user makes changes or updates an existing record. When a user updates an existing record, the new naming convention you specified takes effect.

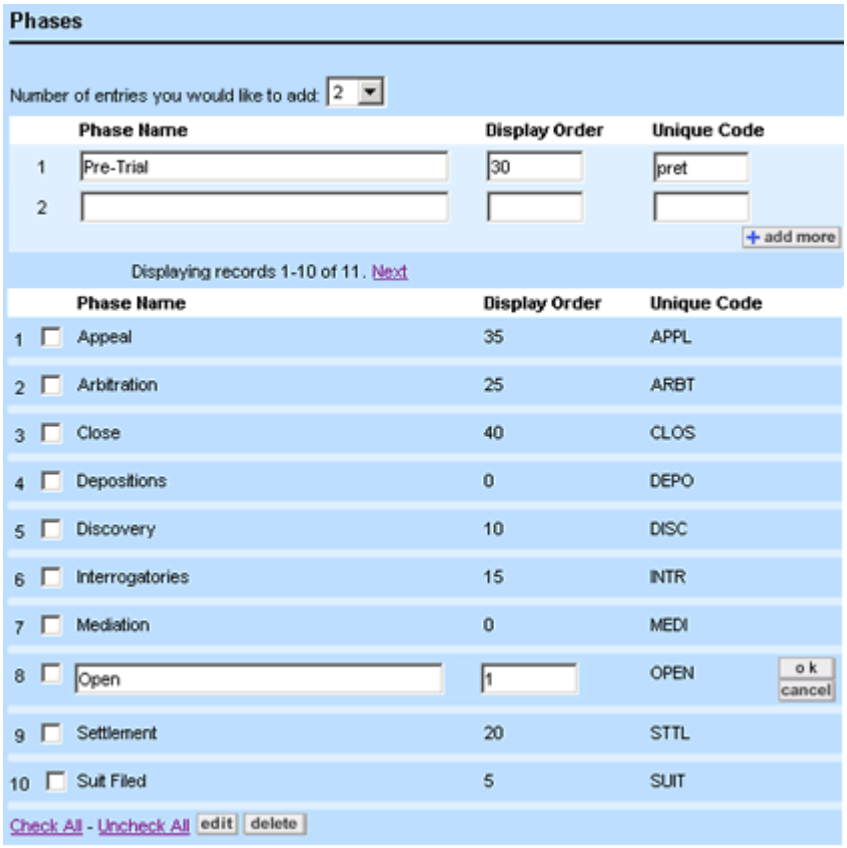
   When the naming pattern is set, the **Name** field in all records of the selected custom object appears as read-only.

1.1.4.4.5 Creating Phases in Custom Objects

All custom object records have their own life cycle consisting of a set of phases. The number of phases and their transitions varies depending on the custom object and your organization's business practices.

You may define the phases for a custom object on the **Phases** tab of its object definition and specify the transition from one phase to another on the **Phase Transitions** tab. For information about setting phase transitions, see Defining Phase Transitions for Custom Objects.

You may create and view all project phases for custom objects on the **Phases** tab. To access this tab, select **Object Definitions** from the **Go to** drop-down list, and then choose the appropriate custom object definition.



Phases Tab

By default the **Phases** tab displays two phases, **Open** and **Close**. They are automatically set as the Initial and Final phases for the project life cycle on the **Phase Transitions** tab. You can always delete these two phases or rename them after you have added other phases.

Phases Tab

Field or control	Description
Phase Name	Type the name of the phase you are adding. The maximum length is 250 alphanumeric characters. This name will be displayed in two places in the end-user interface: the **Phases** tab and on the **General** tab (as the current phase) of the record.
Order	Enter an integer to indicate the order in which you want the name of this phase to be displayed in the drop-down lists on the **Phase Transitions** tab. The phases with the same **Order** are sorted alphabetically and displayed accordingly.

	Tip: *It may be helpful to enter orders in increments of five. If you ever have to add more phases, you will have some leeway as to where in the drop-down list you can place the additional phases.*
Unique Code	Type a 4-character alphanumeric combination to uniquely identify this phase. ***Note:*** *You can type the unique code in uppercase or lowercase, however in the database, it is saved in uppercase.*
Phase List	Displays a list of phases added to the selected custom object. Select or clear the corresponding check-boxes on the left to delete or edit the respective phases.

1.1.4.4.5.1 Adding Phases

Before adding phases, write a complete list of phases you would like to add to a custom object. This helps you ensure that you add all necessary phases and facilitates the process of defining transitions between them.

To create a project phase for a custom object

1. Open the **Phases** tab of the appropriate custom object definition.
2. Select the **Number of entries you would like to add** from the drop-down list.
3. For each data entry row, fill in the information as described in the Phases Tab table.
4. If you want to continue adding phases, click **add more**. Otherwise, click **Save** on the toolbar.
5. Define the desired phase transitions. See Defining Phase Transitions for Custom Objects.

1.1.4.4.5.2 Editing Phases

If you need to change either the name of a phase or its order, you can do so by editing the phase. However, if you need to change its tree position, you must delete the phase first.

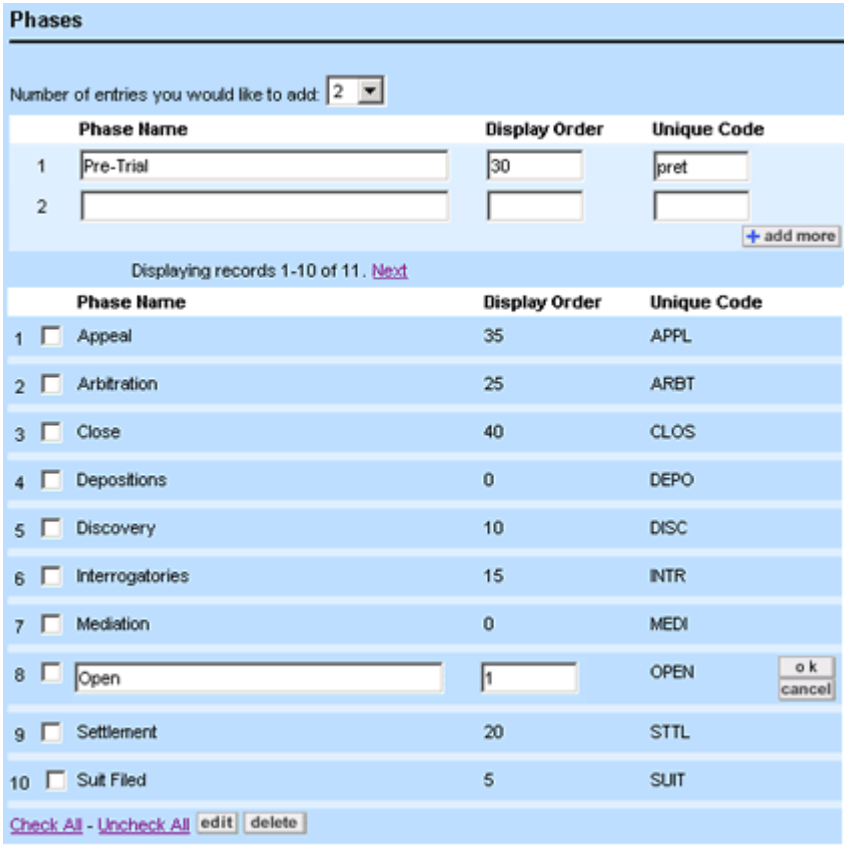
Caution: *If possible, phases should be deleted during the design stage when there are no existing records in the database. Otherwise, loss of data occurs.*

To modify a project phase

1. Open the **Phases** tab of the appropriate custom object definition.

2. Select the check-boxes next to the names of the phases that you want to edit.
3. Click **edit**.

   The phase details that you can modify become editable.



Editable Phases

4. Make the necessary changes to the **Phase Name** or **Order** of the selected items.
5. Click **Save**.

   The changes you made become effective immediately upon saving.

1.1.4.4.5.3 Deleting Phases

If you need to delete a phase or change its tree position, you can do so by deleting the phase.

Caution: *If possible during the design stage, try to delete phases while there are still no existing records in the database. If you delete a phase that is used by records, the phase information is permanently deleted from the records and the phase transition sequence is broken.*

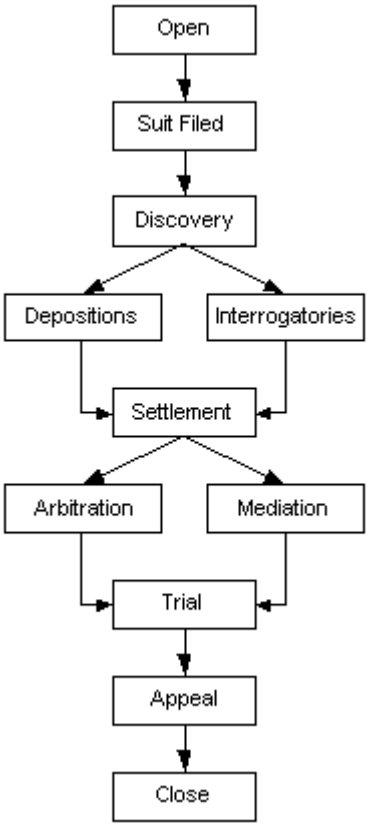
To delete a project phase

1. Open the **Phases** tab of the appropriate custom object definition.
2. Select the check-boxes next to the names of the phases that you want to delete.
3. Click **delete**.
4. When asked to confirm your action, click **OK**.

   The phase information is deleted from the system.

1.1.4.4.6 Defining Phase Transitions for Custom Objects

After you have created the phases necessary for the project life cycle of a custom object, you must specify the way these phases follow each other. Phases may branch into several alternative phases at the same level. For example, the following flowchart shows how the phases shown in the Phase Transitions Tab image may follow each other.



Phase Transition Example

The process of creating a phase transition consists of:

- Naming the phase transition
- Specifying the phase from which the transition occurs
- Specifying the phase to which the transition occurs

- Determining whether the transition is visible to the end user

In the case of a linear transition, you specify the phases as you want them to follow each other, for example, from Open to Suit Filed and from Suit Filed to Discovery.

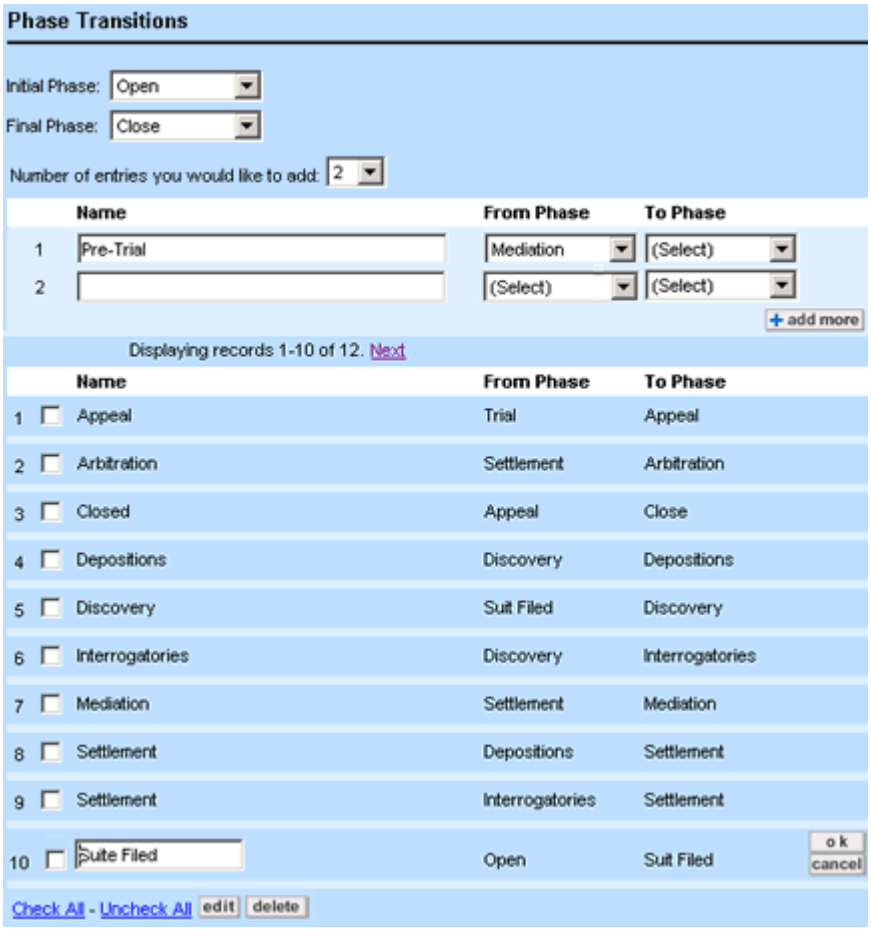
Even if you have only two phases, such as **Open** and **Close**, you must create a phase transition. In this case, select **Open** from the **Initial Phase** field and **Close** from the **Final Phase** field on the **Phase Transitions** tab.

When the phases branch, you must create multiple sequences, such as:

- From Discovery to Depositions
- From Discovery to Interrogatories
- From Depositions to Settlement
- From Interrogatories to Settlement

To access the **Phase Transitions** tab of a custom object definition, select **Object Definitions** from the **Go to** drop-down list, and then choose the appropriate custom object.

The following image shows the **Phase Transitions** tab of a custom object definition. Its phase transitions correspond to those shown in the diagram.



Phase Transitions Tab

The following table describes the items on the **Phase Transitions** tab.

Phase Transitions Tab

Field or control	Description
Initial Phase	Select the phase you want the project life cycle to begin with. The **Initial** phase is automatically linked to the **Opened On** date field in the Project's **General** tab in the end-user interface. ***Note:*** *By default, the* ***Open*** *phase is set as the* ***Initial*** *phase for all projects, and you can change it only when other phases are added on the* ***Phases*** *tab.*
Final Phase	Select the phase you want the project life cycle to end with. The **Final** phase is automatically linked to the **Closed On** date field in the Project's **General** tab in the end-user interface. ***Note:*** *By default, the* ***Close*** *phase is set as the* ***Final*** *phase for all projects, and you can change it only when other phases are added on the* ***Phases*** *tab.*
Name	Type the name of the phase transition as you want it to be displayed for the end user in the **(actions)** drop-down list in the top right-hand corner of the of the Project screen. The maximum length is 250 alphanumeric characters. ***Tip:*** *Type the name of the phase to which you want to go.*
From Phase	Select the phase from which you want the transition to occur.
To Phase	Select the phase to which you want the transition to occur.
Visible	In this check-box, enter a check to make the transition visible to the end user, or clear the box to make the transition non-visible. Non-visible transitions are useful when the transition is always going to be accomplished through rule actions or other automated means, not through manual selection by an end user. By default, all phase transitions are visible.
Phase Transition List	Displays a list of phase transitions set for the selected custom object. Select or clear the corresponding check-boxes on the left to delete or edit the respective phase transitions.

1.1.4.4.6.1 Creating Phase Transitions

Before creating phase transitions in TeamConnect, draw a diagram outlining the sequence in which you want the phases to follow each other. For an example, see Defining Phase Transitions for Custom Objects.

To create a project phase transition

1. Make sure all the necessary phases for the project life cycle are created on the **Phases** tab of the custom object. See Creating Phases in Custom Objects.
2. Open the **Phase Transitions** tab of the appropriate custom object definition.
3. Select the appropriate initial and final phases from the corresponding drop-down lists as described in the Phase Transitions Tab table.
4. Select the **Number of entries you would like to add** from the list.
5. For each data entry row, fill in the information as described in the Phase Transitions Tab table.
6. If you want to continue adding phase transitions, click **add more**. Otherwise, click **Save** on the toolbar.

   The newly added phase transitions appear in the list of phase transitions and the available transitions appear in the actions drop-down list of object records.
7. Test the created phase transitions to ensure no infinite loops or broken sequences were created.

   If the test is successful, the phase transition was successfully created.

1.1.4.4.6.2 Editing Phase Transitions

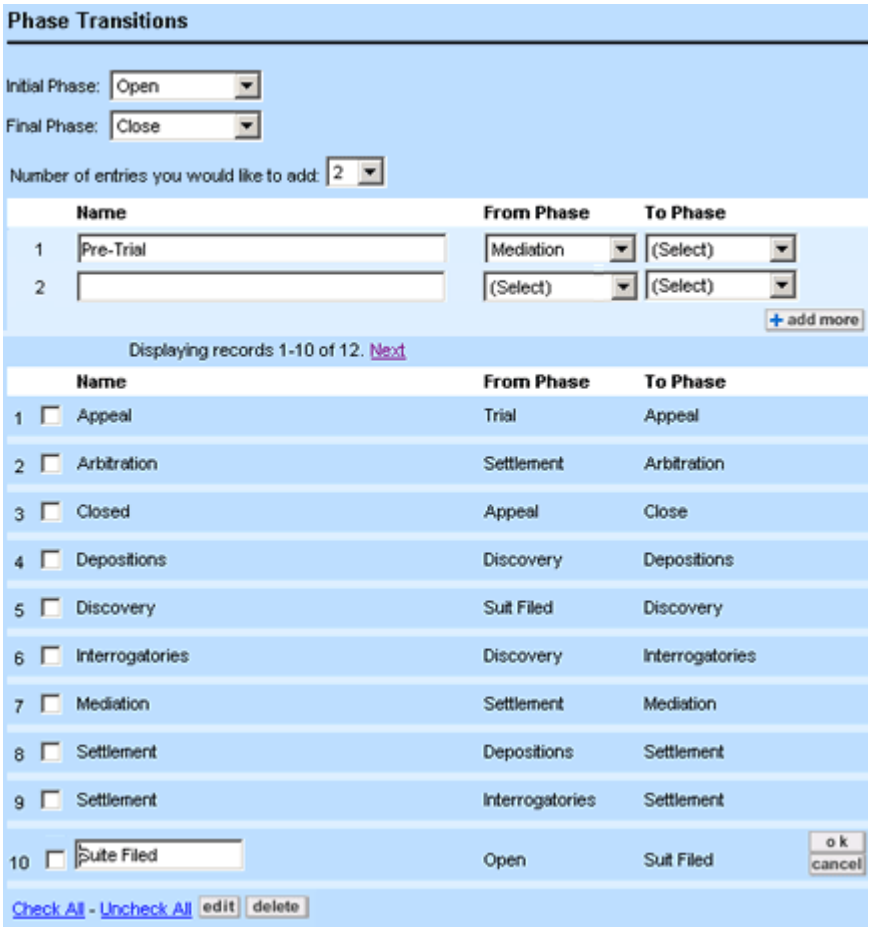
You can edit the name of a phase transition, but you cannot change its **From** and **To** phases. However, you can delete the transition and then recreate it.

Caution: *If possible, phase transitions should be deleted during the design stage when there are no existing records in the database. Otherwise, data will be lost.*

To modify a project phase transition

1. Click the **Phase Transitions** tab of the appropriate custom object definition.
2. Select the check-boxes next to the phase transition that you want to edit.
3. Click **edit**.

   The phase details that you can modify become editable.



Phase Transitions Tab

4. Make the necessary changes to the phase transition **Name** of the selected items.
5. In the **Visible** check-box, enter a check to make the transition visible to the end user, or clear the box to make the transition non-visible. Non-visible transitions are useful when the transition is always going to be accomplished through rule actions or other automated means, not through manual selection by an end user.
6. To change the **Initial Phase** or **Final Phase**, select the desired phase from the drop-down lists at the top of the screen.
7. Click **Save**.
8. Test the phase transitions to ensure no infinite loops or broken sequences exist.

   Once the testing is complete, the phase transition is modified.

1.1.4.4.6.3 Deleting Phase Transitions

If you no longer need a phase transition, you can delete the phase.

Caution: *If possible, delete phase transitions during the design stage when there are no existing records in the database. If you delete a phase transition that is used by existing records, the information is deleted from the records and the phase transition sequence is broken.*

To delete a project phase transition

1. Click the **Phase Transitions** tab of the appropriate custom object definition.
2. Select the check-boxes next the phase transition that you want to delete.
3. Click **delete**.
4. When asked to confirm your action, click **OK**.

   The phase transition information is deleted from the system.
5. Test the remaining phase transitions to ensure that no infinite loops or broken sequences exist.

   The phase transition is deleted.

1.1.4.4.7 Defining Assignee Roles for Custom Objects

Assignee roles are job functions you define for users when they are assigned to a project in the end-user interface. Depending on the custom object, assignee roles may include, for example, associates, supervisors, adjusters for claims, defense attorneys, paralegals, and secretaries for legal matters. In the end-user interface, assignee roles appear in the **Role** drop-down list of the **Assignees** block of project record.

Assignee roles are lookup table items that pertain only to custom object definitions and are defined within them. They have tree positions that identify them in the database. For more details, see Using Categories and Lookup Tables. As a TeamConnect solution developer or administrator, you can insert the desired assignee role in the appropriate table or delete them if necessary through the **Assignee Roles** tab of the respective custom object definition.

1.1.4.4.7.1 Adding Assignee Roles

To add an assignee role to an object

1. Open the **Assignee Roles** tab of the appropriate custom object definition.

Assignee Roles Tab

2. In the **Show roles in node** drop-down list, select the desired hierarchy level where you want to add assignee roles.
3. Select the **Number of entries you would like to add** from the list.
4. For each data entry row, enter the appropriate information in the fields as described in the Assignee Roles Tab table.
5. If you want to continue adding assignee roles, click **add more**. Otherwise, click **Save** on the toolbar.

   The newly added roles appear in the list below the data entry fields and are immediately available in the end-user interface.

The following table describes the items on the **Assignee Roles** tab.

Assignee Roles Tab

Field or control	Description
Show roles in node	Select the desired node (hierarchy level), where you want to add assignee roles.
Role Name	Enter the desired unique name for the role. The maximum length is 50 alphanumeric characters.
Order	Enter an integer to indicate the order in which you want the name of this category to be displayed in the **Role** drop-down list in the end-user

	interface. Roles with the same **Order** are sorted alphabetically and displayed accordingly. ***Tip:*** *It may be helpful to enter orders in increments of five. If you ever have to add more roles, you will have some leeway as to where in the drop-down list you can place the additional roles.*
Tree Position	Enter a 4-character alphanumeric combination to identify the role. See Hierarchical Tree Structure for more details. ***Note:*** *You can enter the tree position in uppercase or lowercase, however, in the database, the tree position is saved in uppercase.*
Role List	Displays a list of roles created for the object. Select or clear the corresponding check-boxes on the left to delete or edit the respective roles.

1.1.4.4.7.2 Editing Assignee Roles

If you need to change the order in which assignee roles appear in the drop-down list in the end-user interface, or their names, you can do so by editing these assignee roles. However, if you need to change their tree positions, you will need to delete the roles first.

Caution: *Roles should be deleted during the design stage when there are no existing records in the database. Otherwise, data will be lost.*

To modify an assignee role

1. Open the **Assignee Roles** tab of the appropriate custom object definition.
2. Select the appropriate node (hierarchy level) where you want to modify assignee roles.
3. Select the check-boxes next to the assignee roles you would like to modify.
4. Click **edit**.

   The corresponding fields of the selected items become editable.

Assignee Roles Tab

5. Make the necessary changes to the **Role Name** or **Order** of the selected items.
6. Click **Save**.

   The assignee role is modified.

1.1.4.4.7.3 Deleting Assignee Roles

If, at the design stage when there are no existing records in the database, you need to delete an assignee role, or change its tree position, you can do so by deleting the assignee role.

Caution: *If the roles you have selected are used in any records, the data entries in these fields will be deleted completely. Also, if you delete a node, all of its sub-levels and their data entries are deleted too.*

To delete an assignee role

1. Open the **Assignee Role** tab of the appropriate custom object definition.
2. Select the appropriate node (hierarchy level) where you want to modify assignee roles.
3. In the role list, select the check-box next to the item you want to delete.
4. Click **delete**.
5. When asked to confirm your action, click **OK**.

6. Click **Save**.

   The assignee role is deleted.

1.1.4.4.8 Customizing Notifications for Object Definitions

Notifications are sent by email to users who have opted to receive them whenever certain actions take place in TeamConnect. The default and custom system notifications can be replaced at the object level with custom notification templates. Customizing notification templates for an object definition as detailed below will supersede any custom notifications in the TeamConnect Admin Settings.

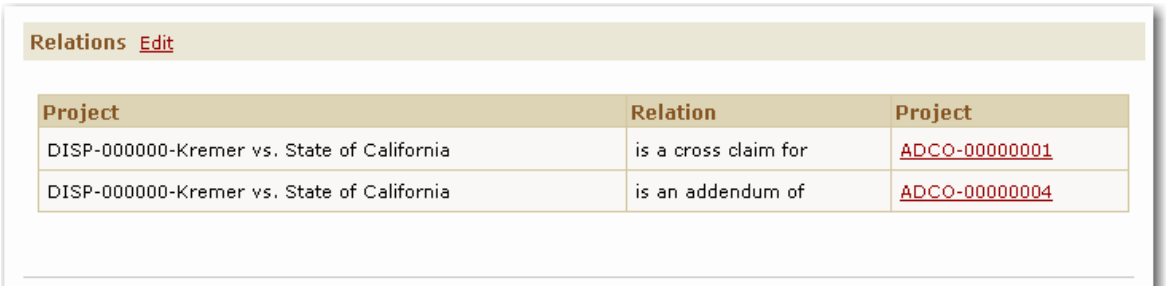
To customize a notification for an object definition

1. Open the **Notifications** tab of the appropriate system or custom object definition.
2. Select the desired notification template from the one of the following drop-down menu options:
   - **Set as an assignee—**The notification that users receive when assigned to a record or task.
   - **Workflow approval request—**The notification that users receive when assigned as the approver of a request.
   - **Workflow approval notice—**The notification that users who complete an action that created an approval request receive when another user approves the request.
   - **Workflow rejection notice—**The notification that users who complete an action that created an approval request receive when another user rejects the request.
   - **Workflow review request—**The notification that users receive when assigned as a reviewer of a request.
   - **Workflow expiration notice—**The notification that approvers receive to remind them that an approval request is about to expire.
   - **Workflow review completion notice—**The notification that users receive when a review they requested is complete.
   - **Set as an attendee—**The notification that users receive when added as an attendee to an appointment.
3. Click **Save**.

1.1.4.5 List Displays in Embedded Objects

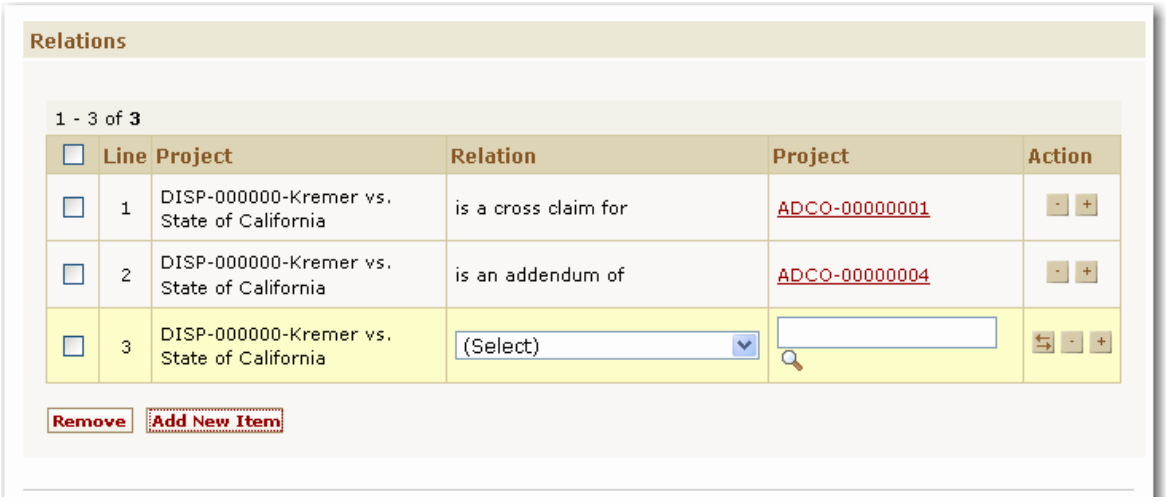
List display is a type of screen that is used to display embedded object records within their parent object record. For example, if Relation is an embedded object in the Dispute custom object definition, the list display you select determines how Relation records appear within the block or section in each Dispute record.

TeamConnect always uses the editable list style to display embedded object records. See the non-editable hyperlinks of the **Project** column in the following image.



Example: Embedded Object as Editable List Block in End-User Interface

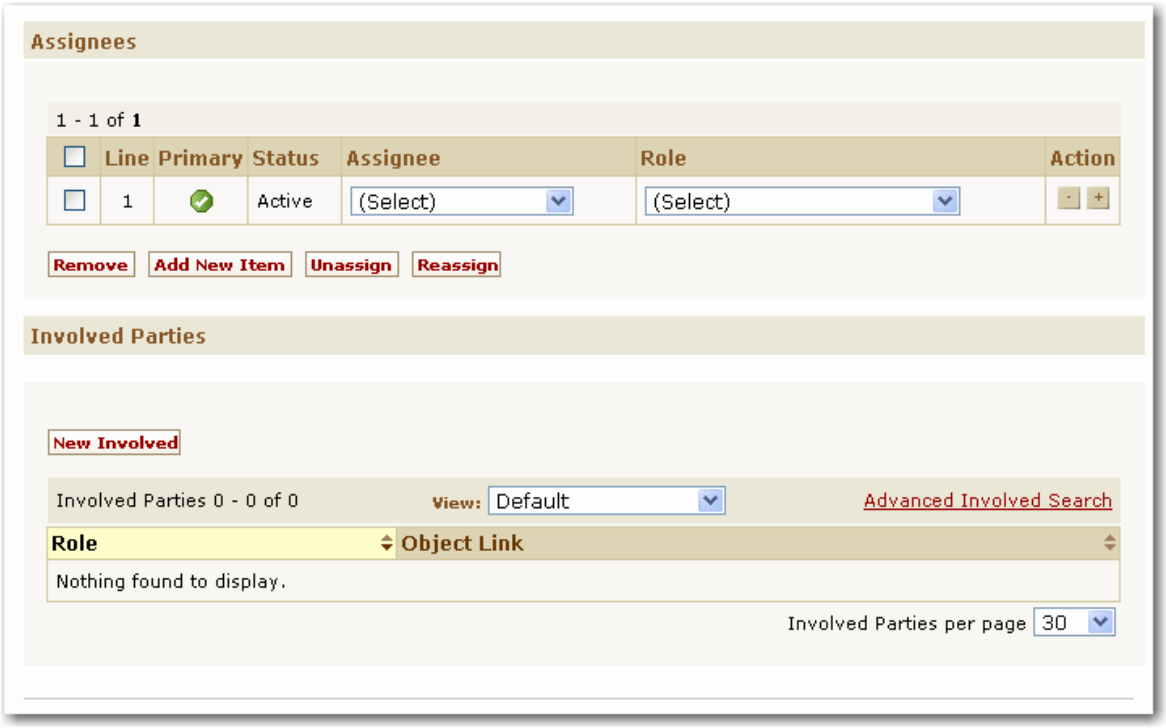
The user clicks either **Edit** (next to **Relations**) or the **Edit** button in the toolbar, and the editable display of the block appears, that is, the list changes from view mode to edit mode. See the object links of the **Project** column in the following image.



Example: Embedded Object List Block in Edit Mode

1.1.4.5.1 Displaying Embedded Records

Embedded object records can be displayed on a tab or within a block in the parent object record. They cannot be displayed in the **All Services** list. When you create embedded objects, you configure their records to be displayed either as search pages with different search views, or as editable lists as shown in the following image.



Example of Embedded Object List Display

You may specify which embedded object record fields appear in the editable list. Users can then edit field values directly from the list and save the changes. That way, if a field value has to be edited in multiple records, users do not have to open each individual record to perform such edits.

1.1.4.5.2 Creating Embedded Objects

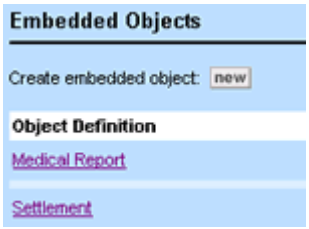
All embedded objects for a custom object can be created through the **Embedded Objects** tab of the respective custom object definition.

Tip: *Creating an embedded object definition is similar to creating a custom object definition. For details, see Creating Custom Objects and the General Tab on Custom Object Definition Screens table.*

Note: *You can convert an embedded object definition to a child object definition if your design requires it. There are several cautions and considerations in doing so. See Converting an Embedded Object Definition to a Child Object Definition*

To create an embedded custom object

1. Open the **Embedded Objects** tab of the appropriate parent custom object definition.



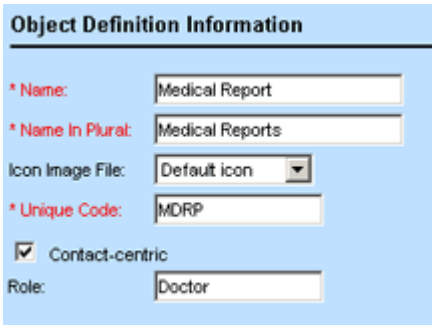
Embedded Objects Tab

2. Click **new**.

   The **General** tab of a new embedded object definition screen appears with empty fields.

3. Enter the following information:

   - Name of embedded object and its plural form
   - Unique code
   - Icon image file



General Tab on Embedded Object Definition Screens

4. If the embedded object definition has a main contact, set it as **Contact-centric** and specify the **Role** for that contact.
5. Click **save**.
6. Click the **List Display** tab and define the list's columns of information in the **Editable List** section of the **List Display** tab. For more information about defining editable lists, see Creating Results Display Columns.
7. Continue defining the embedded object definition's properties as described in Creating Custom Objects.
8. Click save.

   You can access the new embedded object definition either from the parent object definition, or directly from the Object Definition List screen as described in Viewing Object Definitions.

Resulting user interface Changes

Whenever you create an embedded object for a custom object, the following takes place in the corresponding object screens in the Designer:

- A block named after the embedded object is created and becomes available for adding to the custom object views of the parent custom object definition. To add the embedded object's block, open **General** tab of the applicable object view and select it from the **Add the block** field.
- The embedded object becomes available in the **Related Object** drop-down list on the **Filter Display** and **Results Display** tabs of parent object search views.
- The embedded object becomes available on the **Records** tab of the parent object's templates.
- The embedded object becomes available as a related object in all applicable instances of Object Navigator. For details, see Using Object Navigator and Object Model: Read This First plus the additional reference tables this reference points you to.
- The name of the embedded object is added to the list of custom objects on the **Rights** tab of all user and group accounts. This means that in order for the end users to be able to use this object, the rights must be assigned accordingly.

Points To Remember

When creating an embedded object within a custom object, consider the following information:

- You can create multiple embedded objects for a single custom object.

  For example, the Dispute object could have Allegation, Corrective Action, and Damage as its embedded objects.

- In the end-user interface, embedded object records are accessible through a separate page in the record page of the parent record. This page has the same name as the embedded object in the plural.

  The embedded object's block is automatically added to the system view of the parent object. To display embedded object records, you must add the block to the object view used to display the parent object. For more details, see Creating Custom Pages.

- You can set the block for embedded object records to be displayed either as a search screen with different search views, similar to child object records, or as an editable list similar to batch screens used for sub-objects.
- When creating an embedded object, you must assign the appropriate rights for the embedded object to the users and groups, like you do to any custom object.
- Embedded object definitions can be created only from within their respective parent custom object definitions. You cannot create embedded object definitions from the following locations:
  - **Create a new** drop-down list.
  - Object Definition List screen displayed when you choose **Object Definitions** in the **Go to** drop-down list.

  You can, however, see and access the existing embedded object definitions from the Object Definition List.

- End users cannot save an embedded object record unless they save its parent object record.

1.1.4.5.2.1 Contact-Centric Embedded Objects

To create a contact-centric embedded object:

1. Select "contact-centric."

2. Do not add a field for the contact in Custom fields

3. Add the field in list display only. The contact will then be available as a field in the embedded object.

1.1.4.5.3 Defining Display for Embedded Object Records

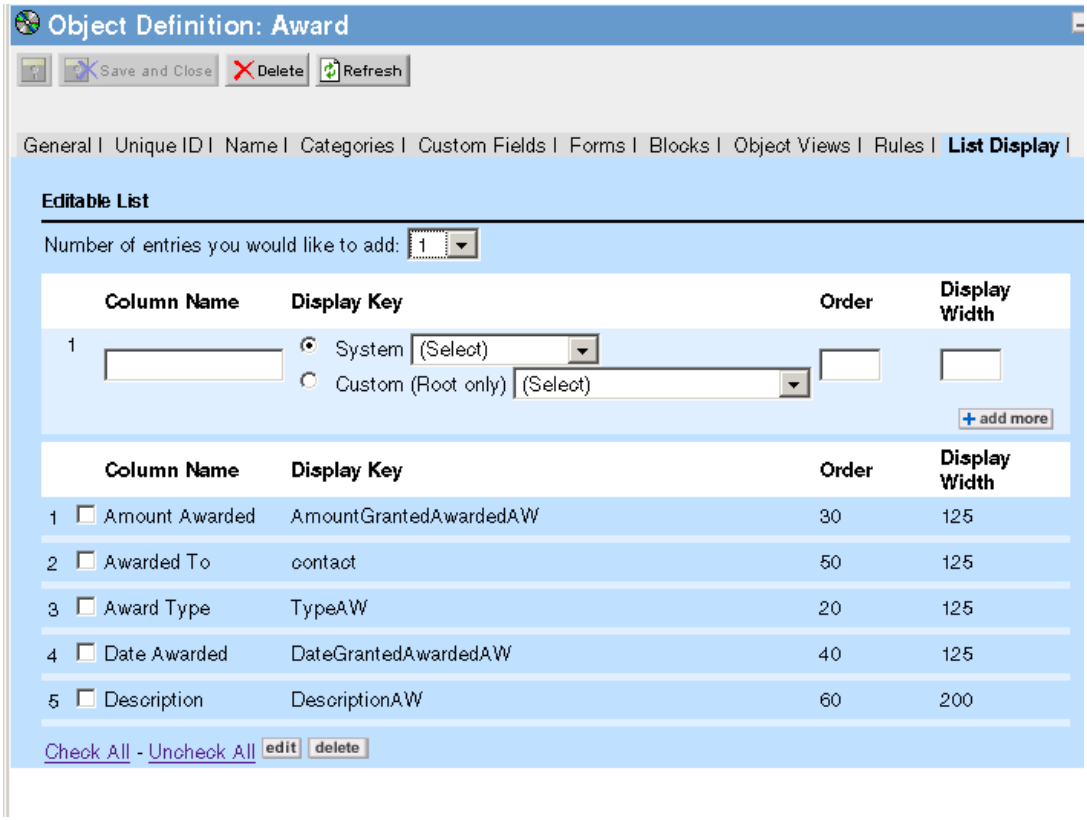
Defining the editable list display for an embedded object is similar to defining the Results display for a search view.

Points To Remember

- If you want users to access records directly from an editable list, include a column with the **Object Link** system display key. When the user clicks the record hyperlink, it displays the record screen of the selected record. See Defining Object Links.
- When adding or editing a record, the column with the object link has no data entry fields or displays a non-editable hyperlink.
- Make sure to add columns for all fields required to create records of the selected embedded object, such as the **Name** field. For details on required fields, see Required Fields.
- You can include custom fields only from the Root category.

To define how embedded object records are displayed within their parent record

1. Open the **List Display** tab of the appropriate embedded object definition.



Editable List Settings for Embedded Objects

2. Define the desired batch screen for the editable list in the same way that you would define the Results display for a search view, using the instructions in Search View Results Display.
3. Save the embedded object definition.
4. Test the created display in the end-user interface.

1.1.4.5.4 Converting Embedded Objects to Child Objects

TeamConnect lets you convert embedded objects to child objects. Child objects and embedded objects are very different, so the conversion changes the way that the converted object displays and performs.

An embedded object is not independent of its parent object and does not have tabs on which to enter information. Embedded objects are only accessible through the parent object.

A child object, though not fully independent of its parent object, has its own set of tabs, such as **Phases**, **Phase Transitions**, **Templates**, and **Security**. When you convert an embedded object to a child object, all of the tabs that are typical to a child object are now available.

Before You Convert

Before you convert an embedded object to a child object, ensure that you have Object Definition Update rights.

What is Not Retained?

Access rights are not retained. You must assign rights to the object as if it were being newly created.

What Are the Prerequisites for Conversion?

You must perform the conversion at a time when no ordinary users are logged in to TeamConnect.

To convert an embedded object definition to a child object definition

1. In the Designer, click **Go to**, and then click **Object Definitions**.
2. In the Object Definitions list, expand the parent or child object definition that contains the embedded object that you want to convert.
3. Click the embedded object.

   The embedded object opens to its **General** tab.
4. Click **Convert to Child object definition**.
5. A confirmation message appears. If there are existing records for the embedded object definition, the record count is shown. If there are a large number of records, you are advised that the confirmation process may require an extended period of time.
6. Click **OK** on the confirmation message if you are sure that you want to convert this object. Wait for the process to complete.

   The former embedded object is now a new child object and displays the tabs that apply to all child object definitions.
7. Click any tab to enter additional information.
8. Save or Save and Close the record.

   The new child object is displayed under its parent object in the Object Definition list view.

Note: *If you convert an object, and then view it before you save it, you may still see properties that applied to the object before you converted it. To ensure a complete conversion, save the converted object before you view it.*

Conversion Results

Note: *If conversion fails due to an application server crash or other major error, it can be re-run later, and will resume with any records that were not completed in the previous run.*

After the conversion, a status message is added to the system log. The new child object now has the following properties:

- A default assignee role named Default with tree key DEFA

- An "All <name of embedded object, such as "~All matters">" search view is set as search across objects
- A default search module with result columns (Opened On and Main Assignee) and criteria (name and number string)
- A templates folders under Root\System\Object Definitions\<name of embedded object, such as "matter">\
- The "Create a Copy" group rights
- All project rule types (user invoke custom action, audit, etc.)
- The **Parent required** and **Do not display in menu** check-boxes are checked on the **General** tab of the object definition
- In the Designer, the following tabs are now available for the new child object:
    - Phases
    - Phase Transitions
    - Assignee Roles
    - Search Views
    - Templates
    - Wizards
    - Embedded Objects
    - History
    - Security
- In the TeamConnect user interface, the following pages are now available for the new child object:
    - Phases
    - Assignees
    - Relations
    - Documents
    - Involved
    - Tasks
    - Appointments
    - Expenses
    - Accounts
    - Workflow
    - History

1.1.4.6 Deleting a Custom Object Definition

You must take some precautions when you are about to delete a custom object definition.

Instances of custom objects become rows in the T_PROJECT database table. If your database has such instances in existence, you must delete those rows before deleting the object definition. Locate the Y_OBJECT_DEFINITION table row that corresponds to the object definition that you wish to delete, and note the value of column APPLICATION_ID in that row. Then run a manual SQL query that deletes T_PROJECT rows where column APPLICATION_ID contains that value.

In addition, if you are running Data Warehouse, ensure that the Data Warehouse synchronization job runs after you have deleted the T_PROJECT rows, and before you actually delete the object definition itself. Data Warehouse tracks any deletions that are done against T_PROJECT and it cannot properly process those deletions if the matching object definition is already gone at the time that the synchronization job runs. After synchronization is complete, you may delete the object definition.

1.1.5 Using Categories and Lookup Tables

Categories and *lookup tables* allow you to organize information that displays on the end-user interface page.

Categories

Categories are hierarchical list items that allow you to organize custom fields of a record into blocks or sections in a given end-user interface page. By adding or deleting categories in a record, end users may display or hide associated blocks of details (custom fields) in that record. Categories also help users to organize records by certain types, for example, employee contact records as opposed to non-employee contact records, or domestic and international accounts.

For another example, you may set up a block so that, if a user adds a **External : Attorney** category to a contact record, a **Contact Vendor Details** block also appears.

You must specify categories when defining:

- Custom Fields
- Blocks
- Access to object records at the category level

Lookup Tables

Lookup tables are database tables that organize, store and quickly access multiple items, such as contacts' roles, types of addresses, phone, fax, and other contact information. In addition to adding, renaming, inactivating, and deleting items in system lookup tables, you may define new custom lookup tables and their contents.

In the end-user interface, lookup tables appear either as drop-down lists (most common) or radio buttons that allow the user to look up necessary information and make an appropriate selection.

TeamConnect's user interface uses several types of list fields that reference lookup table items. For example, users can select the country in which a contact lives from a drop-down list that references the **Country Item** system lookup table.

1.1.5.1 Using Categories

Categories are list items used to organize custom fields into blocks of fields in records. By adding or deleting categories in a record, users display or hide the associated blocks of fields. You may also use categories to organize records by certain types, such as employee contact records as opposed to non-employee contact records.

In custom pages, all custom fields, and custom blocks are tied directly to categories, that is, they are created and displayed by category only. For details, see Custom Fields.

Note: *The word **details** is sometimes used to signify a block of custom fields, in the end-user interface, that are specific to one category. Since many categories may be assigned to a single record, it is also possible for **details** to represent a collection of many such blocks.*

When granting user groups access to information on custom pages, you must give them the rights to the categories for which the custom fields were created.

Points To Remember

Consider the following points when working with categories:

- Based on categories, you may create and display custom fields, and blocks.
- When users delete categories from records, any data in the associated fields is lost.
- If you delete a category from an object definition, the fields associated with the category and any data in the fields are lost.
- Access to all categories in the end-user interface is controlled by the rights assigned to user group accounts.
- The categories for the Involved object are called **Involved Roles** and appear in the **Involved** block of each custom object record.
- When you create a category, you specify a four-character alphanumeric combination to uniquely identify it in the TeamConnect database. For more details, see Category Tree Positions.
- A category name must be unique within its object if it has any custom fields included in Data Warehouse.

1.1.5.1.1 Viewing Categories

You can view lists of categories through the **Categories** tab in a specific object definition.

In the end-user interface, categories appear in the **Categories** page of each object record (except for Tasks and Expenses), where users can select them from the **Category** hierarchical list.

To view an object's categories

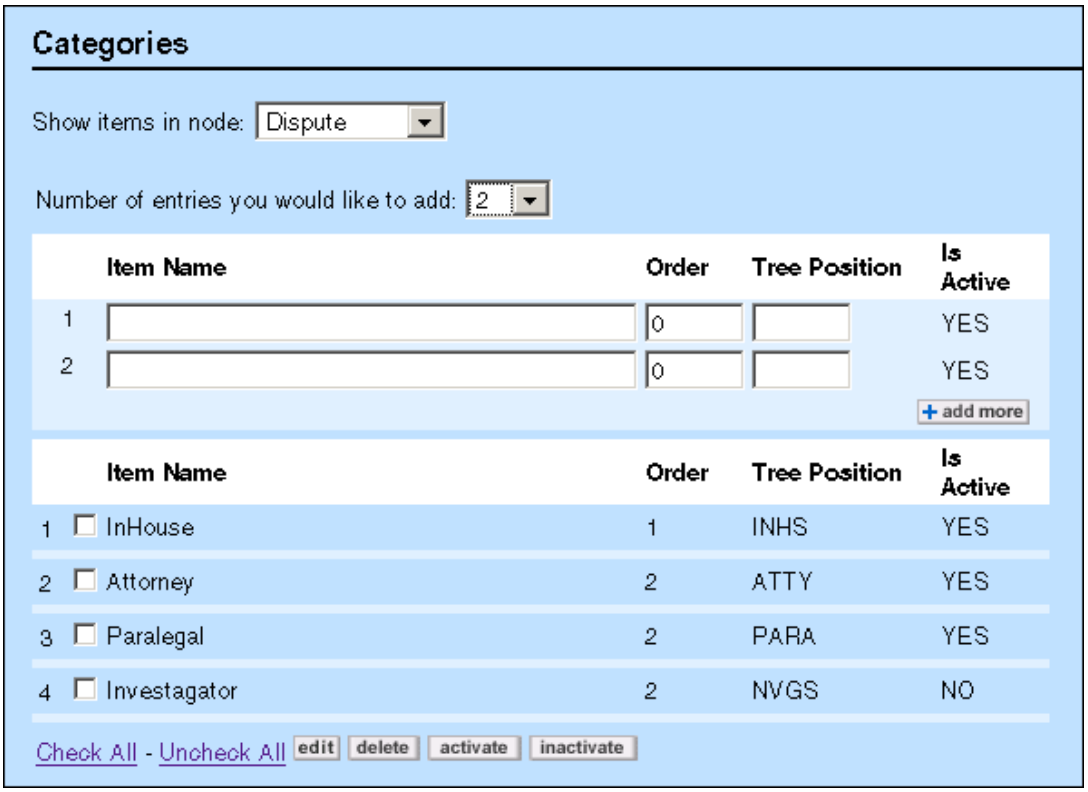
1. In the Designer window, select **Object Definitions** from the **Go to** drop-down list.

   The **Object Definition List** appears.

2. Click the appropriate object definition.

The **General** tab of the object definition appears

3. Click the **Categories** tab.



Categories Tab

4. In the **Show items in node** list, select the appropriate node where you want to view categories.
5. Do one of the following actions:
   a. To add categories, see Adding Object Categories.
   b. To edit categories, see Editing Object Categories.
   c. To inactivate categories, see Inactivating Object Categories.
   d. To delete categories, see Deleting Object Categories.

The following table describes the items in the **Categories** tab.

Categories Tab

Field or control	Description
Show items in node	Select the appropriate node (hierarchy level) where you want to add the category.

Item Name	Enter a name for the category. The maximum length is 50 alphanumeric characters.
Order	Enter an integer to indicate the order in which the category appears in the **Category** drop-down list in the end-user interface. Categories with the same **Order** appear in alphabetical order. ***Tip:*** *It may be helpful to enter orders in increments of five. If you ever have to add more categories, you have leeway as to where you insert them.*
Tree Position	Enter a four-character alphanumeric combination to identify the category. This combination is the category's partial tree position. The full tree position of a category must include the partial tree positions of its parent categories all the way up to the root category. For more details, see Points To Remember. Although you can enter the tree position in uppercase or lowercase, it is saved in uppercase letters in the database.
Category List	Displays a list of categories created for the object. Select or clear the corresponding check-boxes on the left to delete or edit categories.
edit	Click to change the name of the selected category. For more details, see Editing Object Categories.
delete	Click to delete the selected category. ***Caution:*** *Deleting categories can cause the loss of associated data. If you delete a node, all of its sub-levels and their data entries are also removed. If your system is in production, consider inactivating categories instead of deleting them.* For more details, see Deleting Object Categories.
inactivate/ activate	Click to inactivate or activate the selected category. When you inactivate a category, users can see the item in all fields that reference the category, but they cannot select it. Inactivated categories appear differently than active categories. For more information, see Inactivating Object Categories.

1.1.5.1.1.1 Locations of Categories in End-User Interface

Object Category Locations in the End User Interface

Category name	Object type	Section in end-user interface	Field name
Account	System	Categories	Category
Appointment	System	Categories	Category
		General	Default category
Contact	System	Categories	Category
Document	System	Categories	Subject
Expense	System	General	Category
History	System	General	Default category
		Categories	Category
Invoice	System	Categories	Category
Line Item	System	General tab of Invoice > Line Items	Category
Involved (Role)	System	Project > Involved	Default role
		Involved > Roles	Role
Project	Custom	Categories	Category
Task	System	Contact > Rates	Task
		Task > General	Default category

1.1.5.1.1.2 Category Tree Positions

When you create a category on the **Categories** tab of the an object definition, you specify a four-character alphanumeric combination to uniquely identify it in the database. This is its partial tree position in the category hierarchy.

The *full tree position* of a category is a combination of partial tree positions assigned to the category and all of its parent categories, starting with the root category and separated by underscores.

For example, if category B with partial tree position BBBB is created under category A with partial tree position AAAA in the Account object definition, the full tree position of category B is ACCT_AAAA_BBBB.

On the **Categories** tab of the object definition, the full tree position is indicated by the selections you make in the **Show items in the node** drop-down list.

The top-level node is called the Root category. Its full tree position is the same as the "unique code" of the object definition, which are listed in the System Object Unique Codes table and appear in the following locations:

- **General** tab of the respective object definition
- **Object Definition List** screen

You may need a category's full tree position when creating custom blocks, automated actions and qualifiers, document generator templates, XML requests, and so on.

Each Involved and embedded object has its own root category that is independent of the custom object to which it belongs. Its full tree position is the unique code of the respective Involved or embedded object.

The following table lists the unique code (or Root category) of all system objects:

System Object Unique Codes

Object	Unique Code
Account	ACCT
Appointment	APPT
Contact	CONT
Contact Group (Address Book)	CTGR
Document	DOCU
Expense	EXPE
Fee	FEE
History	HIST
Invoice	INVC
Task	TASK
User Account	USER

Group Account	GROU

Likewise, the Root category of custom objects is the same as the object's unique code. For example, if the Dispute object has a unique code DISP, its top-level category would also be named Dispute, and its tree position would be DISP.

1.1.5.1.1.3 System Object Categories

By default the **Categories** tab of system objects displays the following information:

- **Root** as the only available node (hierarchy level), which you cannot delete or inactivate because it is needed to see other categories.

  ***Important: Root** is a concept only. The name of the root node never actually appears as **Root** in the interface. The root node adopts the name of the object itself. Thus the name of the root node for the Contact object is **Contact**, as shown in the figure below.*

  The tree position of the Root category is the same as the unique code assigned to the system object on the General tab of its object definition.

- **Default** category, which you can delete after you add other categories.

As you add categories to a system object, they are added to the **Show items in node** drop-down list and displayed hierarchically. Subcategories are marked by the appropriate indentation. Children of the **Root** category appear at the same level as the **Root** category.

Example: System Object Categories in End-User Interface

1.1.5.1.1.4 Custom Object Categories

By default, the **Categories** tab displays the name of the custom object as the only available node (hierarchy level), which you cannot delete or inactivate because it is needed to see other categories.

The tree position of the Root category is the same as the unique code assigned to the system object on the **General** tab of its object definition.

As you add categories to a custom object, they are added to the **Show items in node** drop-down list and displayed hierarchically. As in system objects, subcategories are marked by the appropriate

indentation in the hierarchy. Children of the root category appear at the same level as the root category.

For example, Claim is the custom object name and root category in the following figure.

Custom Object Categories, as Displayed in End-User Interface

1.1.5.1.2 Adding Object Categories

You define new categories on the **Categories** tab of the respective object definition.

To add a category to an object

1. Open the **Categories** tab of the desired system or custom object definition.
2. In the **Show items in node** list, select the appropriate node where you want to add categories.
3. Select the **Number of entries you would like to add** from the corresponding drop-down list.
4. For each data entry row, fill in the appropriate fields as described in the Categories Tab table.
5. If you want to continue adding categories, click **add more**. Otherwise, click **Save** on the toolbar.

The newly added categories appear in the list below the data entry fields. The categories are available immediately in the end-user interface.

1.1.5.1.3 Editing Object Categories

You can change category names and the order in which they appear in category lists. You cannot change the tree positions of a category unless you first delete the category and then recreate it.

To modify an object category

1. Open the **Categories** tab of the desired system or custom object definition.
2. In the **Show items in node** list, select the appropriate node where you want to modify object categories.
3. Select the check-boxes next to the categories you would like to modify.
4. Click **edit**.

   The corresponding fields of the selected categories become editable.
5. Make the necessary changes to the **Item Name** or **Order** of the selected categories.
6. Click **Save**.

   The changes you made become effective immediately upon saving.

1.1.5.1.4 Inactivating Object Categories

You can inactivate categories so that users are prevented from selecting them from the corresponding records in the user interface. Inactivating categories allows you to prevent the use of unwanted values yet maintain existing data in records that include inactive categories.

Important: *Before inactivating categories, read* *Points To Remember**.*

To inactivate an object category

1. Open the **Categories** tab of the desired system or custom object definition.
2. In the **Show items in node** list, select the appropriate node where you want to inactivate object categories.
3. Select the check-box next to each category you would like to inactivate.
4. Click **inactivate**.

   If the selected categories do not have any child categories, they are inactivated, users cannot select them, and you are finished with this procedure.

   If any of the selected categories have at least one child category, a message box with the following question appears:

   Are you sure you want to inactivate these items?
5. Do one of the following actions:
   a. If you want to inactivate the items and all of their child items, select the check-box next to **Include all child items** and click **ok**.
   b. If you want to inactivate just the items within the selected node but not any of the child items, click **ok**.
   c. If you decide not to inactivate any items, click **cancel**.

      Inactivated categories are indicated in the **Active** column of the **Categories** list of the object definition.

Important: *Update the status of all inactivated categories so that this information is available if you ever need to convert your TeamConnect data.*

Points To Remember

Consider the following points before inactivating categories:

- When you inactivate a category, its associated fields will continue to be displayed in records that already had the category assigned.
- Because an inactive category cannot be selected, its associated fields cannot be displayed in records that did not include the category prior to it being inactivated--even if users add an active child category.
- If an inactive category is assigned to a field via templates, rules, or the XML layer, an exception will be thrown to prevent the creation or update of the record. For example, if a template specifies the default category to be set on Create, but the category is deactivated, the record cannot be created.
- You cannot inactivate categories that are specified as the default category or associated with required custom fields. If you attempt to do so, you will receive an error.
- Inactivated categories are visible to end users in records where they were selected prior to being inactivated. In some circumstances, inactivated categories appear in gray. For example, the **Show details for** field displayed on the **General** tab of some project records will display any inactive categories in gray. If accessibility settings are enabled, **(inactive)** is appended to inactive category names and they are not displayed in gray.
- Users can specify inactivated categories as search criteria and report qualifiers. Inactivated categories appear in gray in search screens or **(inactive)** is appended to the category names.
- Existing records with inactive categories retain the associated custom field values. Users can edit records with previously selected inactivated categories and click **save**.
- If a user deletes an inactive category from a record and then clicks **save**, they cannot revert to the prior value unless a TeamConnect solution developer temporarily activates the category.
- Users can change an inactive category that was specified as the default category of a record (before it was inactivated) to a non-default category. However, once users save the change, they cannot set the inactive category as the default, despite its still being set in the record.
- If you ever need to convert your TeamConnect data, you must first activate all inactive categories and lookup table items. Once the data conversion is completed, you can restore the inactive items manually.
- When you inactivate categories, their status is recorded in the **Active** column of the **Categories** tab in the object definition.
- If a category has parent or child categories, you cannot readily determine their active status without navigating to each level in the hierarchy. Therefore, you should record the status of all inactive categories so that this information is readily available if you ever need to convert your TeamConnect data. Otherwise, you will have to manually search for any inactive categories.
- In the special case of the **Line Item** object definition, you cannot inactivate a category if that category is listed in the **CSM Settings** page, on the pages related to billing codes and to non-

US tax codes, as being part of the default set of codes that are authorized for vendors. This includes task, fee, and expense codes. You must first de-authorize the code in that screen and de-authorize it in each vendor that has that code. Then you can inactivate the category.

1.1.5.1.5 Activating Object Categories

You can activate previously inactivated categories at any time so that users can select them from the corresponding object records in the user interface.

To activate object categories

1. Open the **Categories** tab of the desired system or custom object definition.
2. In the **Show items in node** list, select the appropriate node where you want to activate object categories.
3. Select the check-box next to each category you would like to activate.
4. Click **activate**.

   If the selected categories do not have any child categories, they are activated, users can select them, and you are finished with this procedure.

   If any of the selected categories have at least one child category, a message box with the following question appears:

   Are you sure you want to activate these items?
5. Do one of the following actions:
   a. If you want to activate the categories and all of their child categories, select the check-box next to **Include all child items** and click **ok**.
   b. If you want to activate just the categories within the selected node but not any of the child items, click **ok**.
   c. If you decide not to activate any categories, click **cancel**.

      Activated categories are indicated in the **Active** column of the **Categories** list of the object definition.

Important: *Update the status of all activated categories so that this information is available if you ever need to convert your TeamConnect data.*

1.1.5.1.6 Deleting Object Categories

If you need to delete a category or change its tree position, you can do so by deleting the category. Deleting a category removes all of its sub-categories and corresponding data entries, custom fields, and custom blocks.

Caution: *You should only delete object categories during the design phase, when there are no corresponding object records in the database. Otherwise, loss of data from the associated fields will occur. Alternately, you can inactivate categories.*

To delete an object category

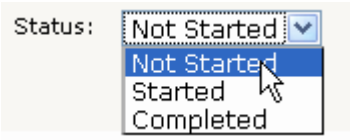
1. Open the **Categories** tab of the desired system or custom object definition.
2. In the **Show items in node** list, select the appropriate node where you want to delete object categories.
3. Select the check-boxes next to the categories you would like to delete.
4. Click **delete**.
5. When asked to confirm your action, click **OK**.
6. Click **Save**.

The selected categories are permanently deleted from the system.

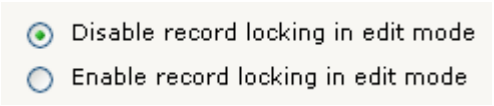
1.1.5.2 Using Lookup Tables

Lookup tables store and list items for users to select. Depending on the system settings and an individual user's Preference settings, lookup tables can be displayed in the following types of fields:

- Drop-down list (most common)
- Option (radio) buttons



Lookup Table Display Example of a Drop-Down List



Lookup Table Display Example of Option Buttons

System lookup tables contain a set of default items. For example, the Address Type lookup table has three items: **Home**, **Business**, and **Other**. You can add to, edit, inactivate, or delete them to fit your design.

You cannot create new system lookup tables. However, you can create custom lookup tables that are specific to your organization. For details, see Creating Custom Pages.

1.1.5.2.1 Types of Lookup Tables

TeamConnect lookup tables can be viewed and modified on the **Lookup Tables** screen, which has separate tabs for the following types of lookup tables:

Lookup Tables Tabs

- System Lookup Tables—Provided by default with TeamConnect, system lookup tables are used in specific areas of TeamConnect. For example, the **Activity Item** table is included for task records. All fields using system lookup tables appear in system blocks. Although you can add or delete items to system lookup tables, you cannot delete the tables. For a complete list of system lookup tables, see the System Lookup Tables table.
- Custom Lookup Tables—Referenced by custom fields to store information specific to your business. You can use custom lookup tables in custom fields for more than one object or belonging to different object categories.

  For more information about custom fields and custom lookup tables, see Creating Custom Pages.
- Multi-currency Table—Stores information about currencies, such as exchange rates and currency symbols. Invoices can be recorded in any currency that is stored in this table.

1.1.5.2.2 System Lookup Tables

System lookup tables are provided with TeamConnect by default for use with specific system objects. You can also create custom lookup tables, as described in Creating Custom Pages.

The following table lists TeamConnect's system lookup tables and their location in the end-user interface. It also indicates which tables can have a hierarchical tree structure.

System Lookup Table Information

Table Name	Object Record	System Block	Field Name	Tree Structure	Unique Code	Table Name in object model
Activity Item	Task	General	Activity		ACTI	LTaskActivityItem
	Invoice	Line items/ Expanded View	Activity		ACTI	LTaskActivityItem
Address Type	Contact	General	Type		ADDR	LContAddressType
Area Item	Appointment	General	Area	**x**	AREA	LApptAreaItem
Contact Relation Type	Contact	Relations	Type of relationship		CONR	LContRelationType
	Involved	Relations	Type of relationship		CONR	LContRelationType

Country Item	Contact	General	Country		COUN	LCountryItem
Email Type	Contact	General	Type		MAIL	LContEmailType
Fax Type	Contact	General	Type		FAXX	LContFaxType
Internet Address Type	Contact	General	Type		INET	LContInetAddressType
Integration Message Queue	This lookup table is deprecated. It provides backward compatibility for existing Integration Message Queue rules.				MESQ	LMessageQueueItem
Invoice Rejection Reason	Invoice	My Approvals pop-up screen	Reason for Reject		INRR	LApphRejectReason
Phone Type	Contact	General	Type		PHON	LContPhoneType
Project Relation Type	Project	Relations	Type of relationship		PRJR	LProjRelationType
Resource Type	Appointment	Resources	Resource		RESO	LApptResourceType
Skill Type	Contact	Skills	Type	**x**	SKIL	LContSkillType
State Type	Contact	General	State/ Province	**x**	STAT	LContState
Territory Type	Contact	Territories	Territory	**x**	TERR	LContTerritoryType

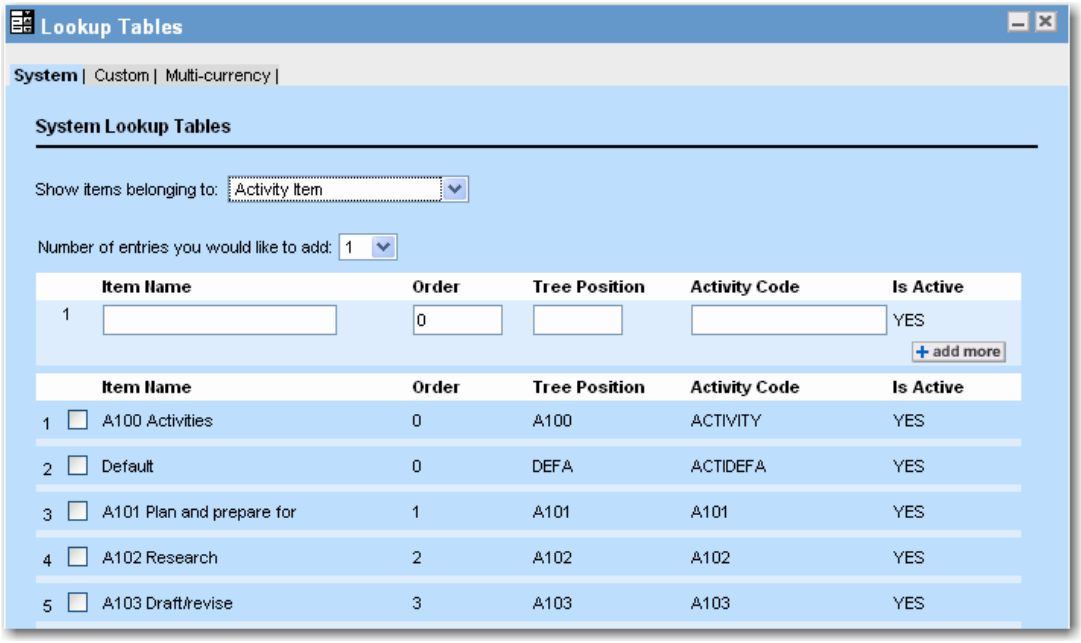
1.1.5.2.2.1 Viewing System Lookup Tables

You can access system lookup tables from the main Designer screen to add, rename, inactivate, or delete items.

To open the system lookup tables screen

1. On main Designer screen, from the **Go to** drop-down list, select **Lookup Tables**.

   The **System Lookup Tables** screen appears, as shown in the following image.



System Lookup Tables Screen

2. Select the appropriate lookup table in the **Show items belonging to** drop-down list.

   If the lookup table has a hierarchical tree structure, another drop-down list appears, labeled **Show items in node**. If necessary, select the appropriate node in the **Show items in node** drop-down list to view the items that are listed within that item.

   The items that have been added to the lookup table (or to the selected item within the lookup table) appear.

The following table describes the items in the System Lookup Tables screen.

System Lookup Tables Screen

Field	Description
Show items belonging to	Select the appropriate system lookup table containing the items you want to view.
Show items in node	Select the appropriate node (hierarchy level) where items are located. ***Note:*** *This field appears only in those lookup tables that can have the hierarchical tree structure.*
Item Name	Enter a name for the item. This name appears to the end-user. The maximum length is 50 alphanumeric characters.

Order	Enter an integer to indicates the order in which it appears to the user. Items with the same order are sorted alphabetically. For details, see Hierarchical Tree Structure. ***Tip:*** *Enter integers in increments of five so that you will have room to insert additional lookup table items if necessary.*
Tree Position	Enter a four-character alphanumeric code that is unique among the items in the lookup table. For details, see Hierarchical Tree Structure. ***Note:*** *While you can enter the tree position in uppercase or lowercase, the tree position is saved in uppercase in the database.*
Lookup Item List	Displays a list of items in the selected node of the lookup table with corresponding check-boxes to delete or edit the item.
edit	Click to change the name of the selected lookup table item. For more details, see Editing System Lookup Tables.
delete	Click to delete the selected lookup table item. ***Caution:*** *Deleting lookup table items can cause the loss of associated data. If you delete a node, all of its sub-levels and their data entries are also removed. If your system is in production, consider inactivating lookup tables instead of deleting them.* For more details, see Deleting Lookup Table Items.
inactivate/ activate	Click to inactivate or activate the selected lookup table item. When you inactivate a lookup table item, users can see the item in all fields that reference the lookup table, but they cannot select it. Inactivated lookup table items appear differently than active items. For more information, see Inactivating Lookup Table Items.

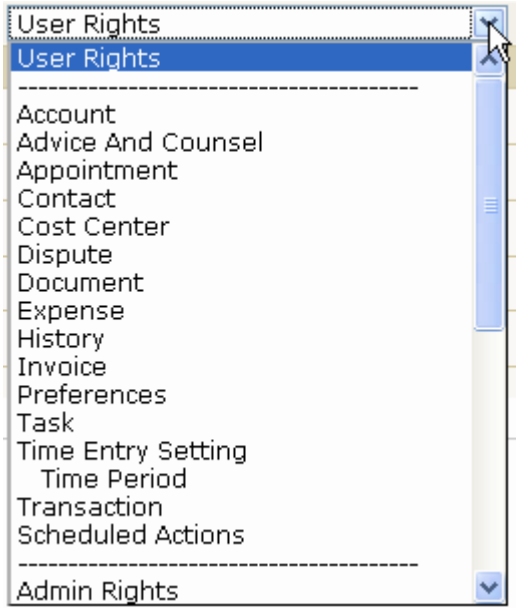
1.1.5.2.2.2 Lookup Table Structure

You may configure the display order of lookup table items and organize them in a hierarchical structure. As you add items to a lookup table, specify their tree positions so they are uniquely identified in the database.

Hierarchical Tree Structure

Many lookup tables consist of one level, so they appear as a flat list of items. However, lookup tables may also have a hierarchical structure with multiple sub-levels. These lookup tables also

display in a drop-down list and appear in the end-user interface with headings and indents. The names of the sub-levels appear in their appropriate positions in the hierarchy, as shown in the following image.



Example: System Lookup Table Hierarchy in Drop-Down List

Levels in the lookup table structure that have their own sub-levels are referred to as Nodes. The tree originates with the Root node, which is the first level and has the following properties:

- Cannot be deleted.
- Not displayed in the end-user interface. Thus, items directly below the Root node appear to end users as first level items.
- In both system and custom lookup tables, the tree position of the Root node is the predefined unique code of the table. For a list of system table unique codes, see the System Lookup Table Information table.

When adding countries and states that you want to appear in the State/Province drop-down lists in the Contact Address block, the following applies:

- If you want a list of states to appear for a country selected in the contact address, be sure that the country is added to the state table with same unique code as the country table and there are states defined as child items of the country item.
- When adding a new country to the Country lookup table, you must also add the new country and its unique code to the state table, and then, if necessary, add the states as child items of the country.

Tree Positions

All items in lookup tables are assigned a unique key called the tree position. In the **Designer** user interface, the tree position appears as four alphanumeric characters in uppercase, such as BUS1.

When you add items to a lookup table, you specify their tree positions so they can be uniquely identified in the database.

In hierarchical tables, each item in a lookup table is identified by the path or "tree" that leads to it within the table's hierarchy. This tree originates with tree position of the Root and continues to the tree position of an item in the hierarchy.

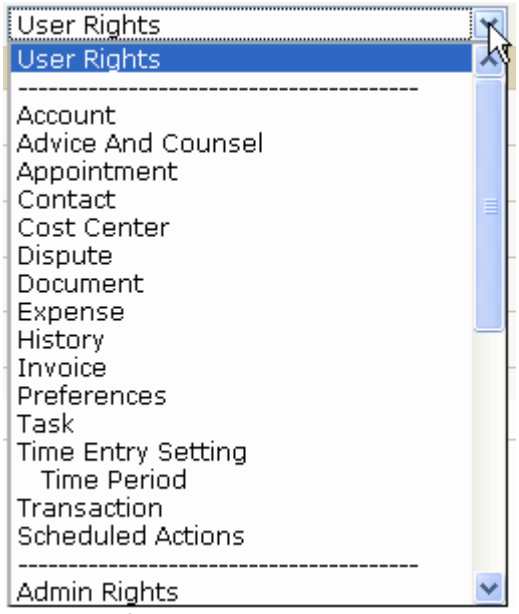
The *full tree position* of a lookup table item is a combination of tree positions assigned to the item and all items above it in the hierarchy, starting with the Root. Each level is separated by underscores.

For example, if an item with tree position CCCC is in the second level node of a lookup table item with the tree position BBBB and the Root's tree position is AAAA, the full tree position of the item is AAAA_BBBB_CCCC.

Order

When you add lookup table items, you can specify an integer to determine the order in which the item appears with respect to the other items in the same hierarchical level (belonging to the same node). Lookup table items with the same order appear alphabetically.

For example, in the following image, the **Order** for **Cost Center** might be 5, because it is the fifth item in its level. **Time Period** might have an **Order** of 1 because it is the first (and only) item in its level.



Example: System Lookup Table Hierarchy in Drop-Down List

1.1.5.2.3 Adding Lookup Table Items

You can add items to system lookup tables to meet your organization's needs. For example, you could add the **is court of** item to the Contact Relation Type lookup table to support relations to contact records of courts.

To add an item to a system lookup table

1. On the **System Lookup Tables** screen, select the appropriate lookup table in the **Show items belonging to** drop-down list.

   If the lookup table has a hierarchical tree structure, the **Show items in node** drop-down list appears. Select the appropriate node in the Show items in node drop-down list.

2. Select the **Number of entries you would like to add** from the corresponding drop-down list.

3. For each data entry row, fill in the appropriate fields as described in the System Lookup Tables Screen table, such as its unique tree position.

4. Click **add more**.

   The newly added item appears in the lookup item list.

1.1.5.2.4 Editing System Lookup Tables

You can edit system lookup table contents as needed by modifying the name or order of items. For example, you could change the name of the **Lap Top** item in the Resource Type lookup table to **Computer** and change its position in the list so that it appears next to the Projector item.

You cannot edit the tree position of an item, but you can add a new item and inactivate or delete the old item.

If renaming the item changes the item's meaning, you should only rename it on a newly installed system. Otherwise, existing data captured using the tree position of the lookup table item will effectively be altered.

To edit a system lookup table item

1. On the **System Lookup Tables** screen, select the appropriate lookup table in the **Show items belonging to** drop-down list.

   If the lookup table has a hierarchical tree structure, select the appropriate node in the **Show items in node** drop-down list.

2. Select the check-box next to each item you would like to change.

3. Click **edit**.

   The corresponding fields of the selected item or items become editable.

4. Make the necessary changes to the **Item Name** and **Order** of the selected items.

5. Click **ok**.

   Your changes are reflected in the lookup item list.

1.1.5.2.5 Inactivating Lookup Table Items

You can inactivate lookup table items so that users are prevented from selecting them from the corresponding fields in the user interface. Inactivating items allows you to prevent the use of unwanted values yet maintain existing data in records that include inactive items.

Important: *Before inactivating lookup table items, read Points To Remember.*

To inactivate lookup table items

1. On the **System Lookup Tables** screen, select the appropriate lookup table in the **Show items belonging to** drop-down list.

   If the lookup table has a hierarchical tree structure, select the appropriate node in the **Show items in node** drop-down list.

2. Select the check-box next to each item you would like to inactivate.

3. Click **inactivate**.

   If the items do not have any child lookup table items, the selected items are inactivated, users cannot select them in any fields, and you are finished with this procedure.

   If any of the selected items have at least one child lookup table item, a message box with the following question appears:

   Are you sure you want to inactivate these items?

4. Do one of the following actions:

   a. If you want to inactivate the items and all of their child items, select the check-box next to **Include all child items** and click **ok**.

   b. If you want to inactivate just the items within the selected node but not any of the child items, click **ok**.

   c. If you decide not to inactivate any items, click **cancel**.

   Inactivated items are indicated in the **Active** column of the lookup item list.

Important: *Record the status of all inactive items so that this information is available if you ever need to convert your TeamConnect data.*

Points To Remember

Consider the following points before inactivating lookup table items:

- If an inactive value is assigned to a field via templates, rules, or the XML layer, an exception will be thrown to prevent the creation or update of the record. For example, if a template specifies an item as the default value of a field to be set on Create, but the item is inactivated, the record cannot be created.
- You cannot inactivate lookup table items used by required custom fields or custom fields that set a default value. If you attempt to do so, you will receive an error.
- Inactivated items are visible to end users in records if they were previously added to a record. Depending on the circumstances, they may be displayed in gray. If accessibility settings are enabled, **(inactive)** is appended to inactive item names and they are not displayed in gray.
- Users can specify inactivated items as search criteria and report qualifiers. Inactivated items appear in gray or **(inactive)** is appended to the item names.

- Existing records with inactive items retain the inactive values. Users can edit records with previously selected inactivated items and click **save**.
- If a user changes the value of a field that specified an inactive item and then clicks **save**, they cannot revert to the prior inactive value unless a TeamConnect solution developer temporarily activates the item.
- If you ever need to convert your TeamConnect data, you must first activate all inactive categories and lookup table items. Once the data conversion is completed, you can restore the inactive items manually.
- When you inactivate lookup table items, their status is recorded in the **Active** column of the lookup table screen.
- If an item has parent or child items, you cannot readily determine their active status without navigating to each level in the hierarchy. Therefore, you should record the status of all inactive items so that this information is readily available if you ever need to convert your TeamConnect data. Otherwise, you will have to manually search for any inactive items.

1.1.5.2.6 Activating Lookup Table Items

You can activate previously inactivated lookup table items so that users can select them from the corresponding fields in the user interface.

To activate lookup table items

1. On the **System Lookup Tables** screen, select the appropriate lookup table in the **Show items belonging to** drop-down list.

   If the lookup table has a hierarchical tree structure, select the appropriate node in the **Show items in node** drop-down list.

2. Select the check-box next to each item you would like to activate.
3. Click **activate**.

   If the items do not have any child lookup table items, the selected items are activated, users can select them from corresponding fields, and you are finished with this procedure.

   If any of the selected items have at least one child lookup table item, a message box with the following question appears:

   Are you sure you want to activate these items?

4. Do one of the following actions:

   a. If you want to activate the items and all of their child items, select the check-box next to **Include all child items** and click **ok**.

   b. If you want to activate just the items within the selected node but not any of the child items, click **ok**.

   c. If you decide not to activate any items, click **cancel**.

   Activated items are indicated in the **Active** column of the lookup item list.

Important: *Update the status of all activated lookup table items so that this information is available if you ever need to do any kind of conversion of your TeamConnect data.*

1.1.5.2.7 Deleting Lookup Table Items

System lookup tables themselves cannot be deleted. However, you can delete items in a system lookup table if they are not needed. TeamConnect automatically checks to make sure that each item is not being referenced by records. If an item is being used, you are prompted to confirm its deletion.

You must consider the impact of deleting lookup table items—especially on an existing system that has records using the item. If the tree position of the item is used by rules or reports, you must consider the impact of deleting it and make any necessary modifications. You may be able to inactivate or rename the item rather than delete it.

Important: *If you delete an item, all record data referencing it is lost. If you delete a node, all of its sub-levels and their data entries are also removed.*

To delete an item from a system lookup table

1. On the **System Lookup Tables** screen, select the appropriate lookup table in the **Show items belonging to** drop-down list.

   If the lookup table has a hierarchical tree structure, another drop-down list appears, **Show items in node**. Select the appropriate node in the **Show items in node** drop-down list.

2. Select the check-box next to each item in the list that you would like to delete.

3. Click **delete**.

   The items you deleted are immediately removed from the lookup item list and all screens where the lookup table is accessible in the end-user interface.

1.1.5.2.8 Custom Lookup Tables

Like system lookup tables, custom lookup tables can be hierarchical in nature, automatically start with a Root node, and appear in the order specified for each item in the table.

Each custom lookup table is identified by its four-character alphanumeric unique code, which is assigned when you create the table.

Each item in a custom lookup table is assigned a *partial tree position*, a four-character alphanumeric combination, when you add it to the table.

To identify the option selected as a value in a custom field of type List when using the API, XML Layer, or other back end functionality, you must use the *full tree position* of the corresponding item in the custom lookup table associated with the field.

It starts with the unique code of the table + ROOT and includes the partial tree position of the parent items (if there are any) and the selected item itself, in UPPERCASE and separated by underscores, for example, OCCU_ROOT_HURE_HRAS.

1.1.5.2.8.1 Viewing Custom Lookup Tables

The Custom Lookup Tables screen allows you to view existing custom lookup tables. If you need to create new ones, use this procedure to access the screen, and then see Adding Custom Lookup Tables.

To open the custom lookup tables screen

1. In the Designer window, in the **Go to** drop-down list, select **Lookup Tables**.
2. Click the **Custom** tab.

   The **Custom Lookup Tables** screen appears.
3. Select the appropriate user-defined lookup table in the **Show items belonging to** drop-down list.
4. Select the appropriate node in the **in node** drop-down list, if necessary.

A list of items available for the selected lookup table or its selected node appears below the drop-down lists.

The following table describes the items in the custom lookup tables screen.

Custom Lookup Tables Screen

Field or button	Description
New lookup table name	***Important:*** *Use this field only if you want to create a lookup table.* Enter a name for a new custom lookup table. Maximum length is 50 alphanumeric characters. However, Data Warehouse only supports a maximum of 35 characters.
Unique Code	***Important:*** *Use this field only if you want to create a lookup table.* Enter a four-character alphanumeric code that is unique across all custom lookup tables. ***Note:*** *You can enter the unique code in uppercase or lowercase, however, in the database, it is saved in uppercase.*
Add Table	Click to add the new table to the custom lookup tables listed in the **Show items belonging to** drop-down list.
Show items belonging to	Select the custom lookup table where you want to view, add, edit, or delete items.

in node	Select the node (hierarchy level), if necessary, to locate the appropriate item or items.
Delete Table	Click to delete the selected table in the **Show items belonging to** drop-down list. See Deleting Custom Lookup Tables.
Item Name	Enter a name for the option as it should be displayed in the field. Maximum length is 50 alphanumeric characters.
Order	Enter an integer, indicating the **Order** for the item in the list. The items with the same **Order** are sorted alphabetically and displayed accordingly. ***Tip:*** *It may be helpful to enter orders in increments of five. If you ever have to add more options, you will have some leeway as to where in the list you can place the additional items.*
Tree Position	Enter a four-character alphanumeric combination that uniquely identifies the lookup item within the node in which you are adding it. ***Note:*** *You can enter the tree position in uppercase or lowercase, however, in the database, it is saved in uppercase.*
Is Active	• **YES—**The lookup table item is active • **NO—**The lookup table item is inactive and cannot be selected by end users. For more details, see Inactivating Custom Lookup Table Items.
Lookup Item List	Displays a list of lookup items in the selected node of the table. Select or clear the corresponding check-boxes on the left to delete or edit lookup items.

1.1.5.2.8.2 Adding Custom Lookup Tables

You can add as many custom lookup tables for custom fields of type List as you need. Remember, however, that you can re-use the same table for multiple custom fields, for example, if you need to add a field with the same options under different categories or even in different object definitions.

Tip: *Always make sure that the names you assign to your custom tables are self-explanatory and indicative of the items they contain or for which fields they are created.*

To add a new custom lookup table

1. On the **Custom Lookup Tables** screen, in the **New lookup table name** field, enter a name for the table.

   The table name can be 50 characters maximum in length.

2. In the **Unique Code** field, type four alphanumeric characters for a unique identifier for the table.

3. Click **Add Table**.

   The newly added table appears alphabetically in the **Show items belonging to** drop-down list.

4. Add the desired items to the table. For details, see Adding Items to Custom Lookup Tables.

   The custom lookup table is created and available for use in custom fields of type List.

1.1.5.2.8.3 Deleting Custom Lookup Tables

You can delete custom lookup tables that are no longer necessary for custom fields. However, you will not be able to delete a custom lookup table if it is used by a custom field or if any of its items is used in a project auto naming pattern in custom object definitions.

To delete a custom lookup table

1. On the **Custom Lookup Tables** screen, select the custom lookup table you would like to delete in the **Show items belonging to** drop-down list.

2. Click **Delete Table**.

3. When asked to confirm your action, click **OK**.

If the table is already being used in records, you will receive the appropriate message, otherwise the table with all of its items is deleted from the database.

1.1.5.2.8.4 Adding Items to Custom Lookup Tables

You need to add items to custom lookup tables to maintain the appropriate list of selections for your users, according to your business needs.

To add new items to a custom lookup table

1. On the **Custom Lookup Tables** screen, in the **Show items belonging to** drop-down list, select the custom lookup table to which you need to add items.

2. In the **in node** drop-down list, select the appropriate level where the items should be.

3. Select the **Number of entries you would like to add**.

4. For each data entry row, enter the following information:

   - Name of the item as it should be displayed in the field.

- Its display order.
- A four-character alphanumeric combination that is unique in the selected node.
- If necessary, for more specific details, see the Custom Lookup Tables Screen table.

5. Click **add more**.

   The newly added item appears in the lookup item list.

1.1.5.2.8.5 Editing Custom Lookup Table Items

You can edit custom lookup table contents when you want to display them in a different order in the user interface, or if you want to change the name that appears for each item.

You cannot edit the tree position of a lookup item. During the design stage, when no records have been created yet, you can add a new item and then delete the item that has the incorrect tree position.

To edit an item in a custom lookup table

1. On the **Custom Lookup Tables** screen, in the **Show items belonging to** drop-down list, select the custom lookup table where you need to edit items.
2. In the **In Node** drop-down list, select the appropriate level where you want to make changes.
3. Select the check-box next to each item you need to change.
4. Click **Edit**.

   The corresponding fields of the selected items become editable.
5. Make the necessary changes to the **Item Name** and **Order** of the selected items.
6. Click **OK**.

   Your changes are reflected in the lookup item list.

1.1.5.2.8.6 Inactivating Custom Lookup Table Items

You can inactivate lookup table items so that users are prevented from selecting them from the corresponding fields in the user interface. Inactivating items allows you to prevent the use of unwanted values yet maintain existing data in records that include inactive items.

Important: *Before inactivating lookup table items, read Points To Remember.*

To inactivate lookup table items

1. On the **Custom Lookup Tables** screen, select the appropriate lookup table in the **Show items belonging to** drop-down list.

If the lookup table has a hierarchical tree structure, select the appropriate node in the **Show items in node** drop-down list.

2. Select the check-box next to each item you would like to inactivate.
3. Click **Inactivate**.

   If the items do not have any child lookup table items, the selected items are inactivated, users cannot select them in any fields, and you are finished with this procedure.

   If any of the selected items have at least one child lookup table item, a message box with the following question appears:

   Are you sure you want to inactivate this item?

4. Do one of the following actions:

   a. If you want to inactivate the items and all of their child items, select the check-box next to **Include all child items** and click **ok**.

   b. If you want to inactivate just the items within the selected node but not any of the child items, clear the check-box next to **Include all child items** and click **ok**.

   c. If you decide not to inactivate any items, click **Cancel**.

      Inactivated items are indicated in the **Active** column of the lookup item list.

Important: *Record the status of all inactive items so that this information is available if you ever need to convert your TeamConnect data.*

Points To Remember

Consider the following points before inactivating lookup table items:

- If an inactive value is assigned to a field via templates, rules, or the XML layer, an exception will be thrown to prevent the creation or update of the record. For example, if a template specifies an item as the default value of a field to be set on Create, but the item is inactivated, the record cannot be created.
- You cannot inactivate lookup table items used by required custom fields or custom fields that set a default value. If you attempt to do so, you will receive an error.
- Inactivated items are visible to end users in records if they were previously added to a record. Depending on the circumstances, they may be displayed in gray. If accessibility settings are enabled, **(inactive)** is appended to inactive item names and they are not displayed in gray.
- Users can specify inactivated items as search criteria and report qualifiers. Inactivated items appear in gray or **(inactive)** is appended to the item names.
- Existing records with inactive items retain the inactive values. Users can edit records with previously selected inactivated items and click **save**.
- If a user changes the value of a field that specified an inactive item and then clicks **save**, they cannot revert to the prior inactive value unless a TeamConnect solution developer temporarily activates the item.

- If you ever need to convert your TeamConnect data, you must first activate all inactive categories and lookup table items. Once the data conversion is completed, you can restore the inactive items manually.
- When you inactivate lookup table items, their status is recorded in the **Active** column of the lookup table screen.

If an item has parent or child items, you cannot readily determine their active status without navigating to each level in the hierarchy. Therefore, you should record the status of all inactive items so that this information is readily available if you ever need to convert your TeamConnect data. Otherwise, you will have to manually search for any inactive items.

1.1.5.2.8.7 Activating Custom Lookup Table Items

You can activate previously inactivated lookup table items so that users can select them from the corresponding fields in the user interface.

To activate lookup table items

1. On the **Custom Lookup Tables** screen, select the appropriate lookup table in the **Show items belonging to** drop-down list.

   If the lookup table has a hierarchical tree structure, select the appropriate node in the **Show items in node** drop-down list.
2. Select the check-box next to each item you want to activate.
3. Click **activate**.

   If the items do not have any child lookup table items, the selected items are activated, users can select them from corresponding fields, and you are finished with this procedure.

   If any of the selected items has at least one child lookup table item, a message box with the following question appears:

   Are you sure you want to activate this item?
4. Do one of the following actions:
   a. If you want to activate the items and all of their child items, select the check-box next to **Include all child items** and click **ok**.
   b. If you want to activate just the items within the selected node but not any of the child items, clear the check-box next to **Include all child items** and click **ok**.
   c. If you decide not to activate any items, click **cancel**.

      Activated items are indicated in the **Active** column of the lookup item list.

Important: *Update the status of all activated lookup table items so that this information is available if you ever need to convert your TeamConnect data.*

1.1.5.2.8.8 Deleting Custom Lookup Table Items

You can delete items in a custom lookup table when they are no longer needed. For example, in the design stage of implementation, you might need to clean up the contents of a custom lookup table to have the associated field display the appropriate selections.

TeamConnect automatically checks to make sure that each lookup item is not being used in records. If an item is being used as data in a record, you are notified and asked to confirm your decision to delete it.

Caution: *Deleting a lookup item also deletes the item's data entries in records, all of its child items, and the child items' data entries.*

To delete an item from a custom lookup table

1. On the **Custom Lookup Tables** screen, in the **Show items belonging to** drop-down list, select the custom lookup table where you need to delete items.
2. In the **in node** drop-down list, select the appropriate level where you want to make changes.
3. Select the check-box next to each item you want to delete.
4. Click **delete**.
5. When asked to confirm your action, click **OK**.
6. If the item or one of its child items is being used as data in records, another confirmation is required. Click **OK**.

   The item and all of its children, if there are any, are deleted along with their record entries.

1.1.6 Creating Custom Pages

You may define the following components of TeamConnect custom pages to determine the appearance of your organization's record pages:

- **Custom fields**—Fields that you create for specific categories of system and custom objects to meet your organization's needs. You specify the label that appears in the end-user interface and what type of value the field holds, such as Memo Text or Check-Box. You may add custom fields to custom blocks.
- **Blocks**—A group of system and custom fields included into a single XML file for custom blocks or Java Server Page (JSP) file for system blocks. You may add blocks to object views and wizard pages.
- **Object views**—Layout of record pages, blocks, and other user interface elements on the page that you may customize and assign to different user groups. You may create multiple object views for each object definition, so different groups see only the information relevant to them.
- **Custom tools**—You may create tools specific to your organization to help automate a task. You may define how the page appears and assign rights to the user groups that need the tool. For details on how to create custom tools, see Creating Custom Tools.

TeamConnect provides system fields and default object views for all system objects. When you create a custom object, it includes a predefined object view. Depending on the needs of your

organization, creating custom blocks and object views may give you more flexibility in page layout and greater control of user access to record information.

Use the Screen Designer tool to create blocks.

Note: *The Designer user interface refers to most interface windows in the browser as "screens." The end-user interface refers to its own interface windows as "pages." This document refers to Designer interface windows as "screens" and the end-user equivalents as "pages."*

Important: *Keep in mind that the Designer user interface may often refer to end-user interface windows (pages) as "screens."*

This chapter details the following sequence of creating custom pages:

- Creating Custom Fields
- Creating Object Views

1.1.6.1 Custom and System Views

Each object definition in TeamConnect has its own system fields organized into system blocks and system views. By default, system views display generic information to all user groups--unless you create and assign custom views.

The following table outlines the major differences between custom and system views in object definitions.

System vs. Custom Views

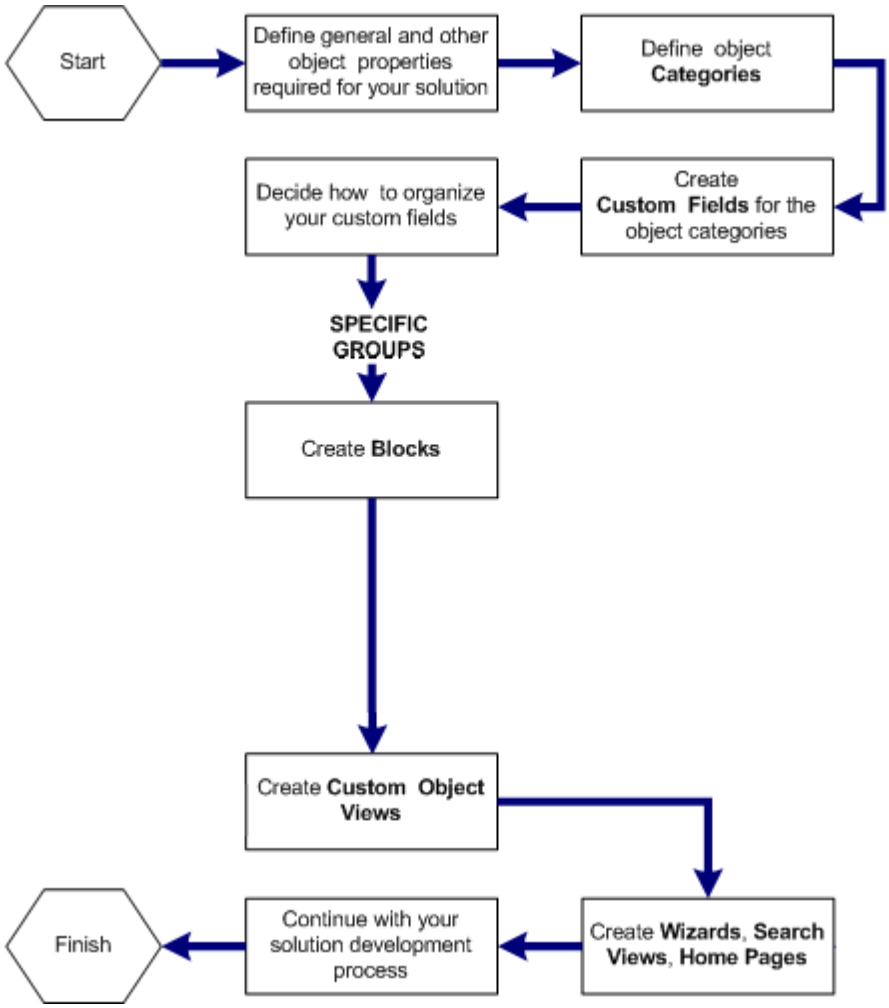
Custom views	System views
You define your own object views.	You accept the predefined object views.
The page layout is alterable. For example, you can reposition pages or blocks.	The page layout is fixed.
You may use custom fields with the organization-specific information to create blocks.	Do not use custom fields with the organization-specific information to create blocks.
You may restrict access to your custom information by creating different object views for different groups using different blocks.	You may restrict access to your custom information only by category, through assigning different rights. Otherwise, all users see the same blocks.

While system views may meet your organization's needs for certain system objects, such as Appointment or Contact, you probably need to create custom views for custom objects.

1.1.6.2 Process Overview

The following flowchart is a snippet of the general customization process. It details the steps and decisions you have to make while organizing your custom fields for display in record pages.

When your custom fields are organized, you may proceed to creating the rest of custom pages, such as wizards, search views, and home pages.



Custom Page Creation Process

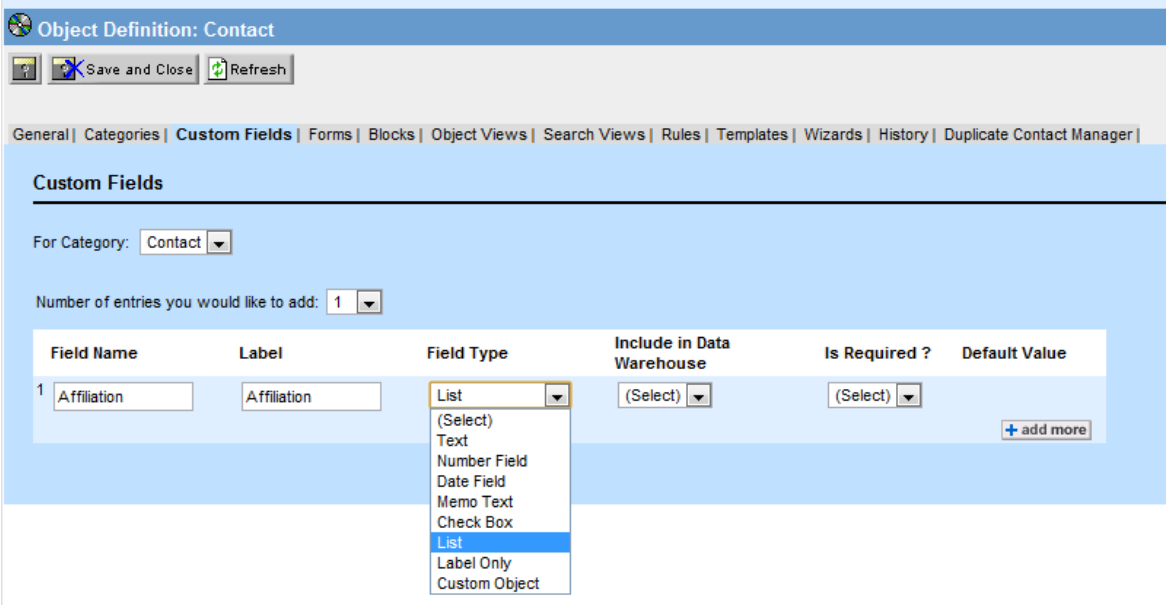
1.1.6.3 Custom Fields

You may create custom fields to capture information specific to your organization's business needs. The default installation of TeamConnect is configured to capture generic information that most organizations need, such as record name, number, date when created, user who created it, and so on. However, your organization might need to include information that is specific to your industry and business methods.

For example, if your organization shipped packages, you might need to store the number of packages shipped to a particular client, the total weight of the packages, and the type of delivery. The default installation of TeamConnect does not include such fields, but you could create custom fields for the purpose. To define each of these fields, you need to know the following information:

- What kind of information it should contain so that you can label it accordingly, such as **Total Shipping Weight**.
- What field type it must be to store the necessary information properly, such as Number, List, or Text fields.

After you enter this information on the **Custom Fields** tab of the appropriate object definition, the users with the appropriate rights see the fields on their pages, as shown in the following images.



Custom Fields in the Designer Interface

To have your custom fields displayed on the page, you have to include them in blocks. These elements provide a layout for custom fields on the page and allow you to add certain formatting attributes to their tags.

1.1.6.3.1 Where To Use Custom Fields

You create custom fields to capture and display your organization-specific information and then assign the appropriate object views. The following table lists the areas and features that use custom fields to capture information or to implement the desired workflow. The table also indicates where to find additional information.

Where Custom Fields May Be Used in TeamConnect

Feature	Description	For details, see
Wizards	Custom fields may be added as individual components or within blocks to gather initial information when a record is being created	Creating Wizards
Search Views	Custom fields may be used as search criteria and search results display keys	Creating Search Views

Rules	Custom fields may be used as qualifiers in user interface rules, as well as in actions in custom action rules	Using Rules
Auto Generated Letters and Documents	Custom field values may be used by the Document Generator to automatically create letters and other documents	Administrator Help
XML Layer	Custom field values may be used for data conversion and integration with other applications	*XML Layer Reference*

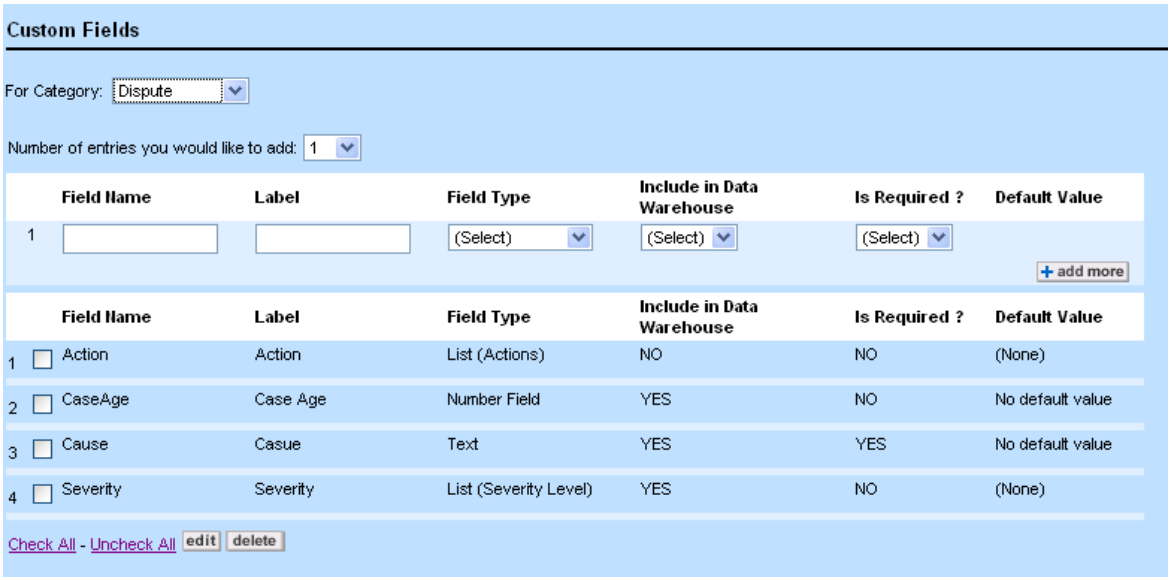
Points To Remember

Consider the following points about custom fields:

- Custom fields are category-specific. That is, they can be created and accessed only through their respective categories.
- Access and display of custom fields is controlled by the user's rights to each category and its respective custom fields and whether the category is active.
- If you inactivate categories, users are prevented from selecting them from the corresponding object's records and the associated blocks of custom fields are not displayed. For more information about inactivating categories, see Inactivating Object Categories.
- To reference a custom field, in most cases, you need to know its field name (not label), the full tree position of the category under which it is created, and the field type.
- In the database, object model, XML Layer, and API, tables, tags, and interfaces that deal with custom field definition or value information typically have the word "Detail" as part of their name (for example, `Y_OBJ_DETAIL_FIELD`, `<Detail/>`, or `EnterpriseEntity`). Hence, you may also come across references to custom fields as "detail fields."

1.1.6.3.2 Custom Fields Tab in Object Definitions

All custom fields for an object may be created, viewed, modified, or deleted using the **Custom Fields** tab of the corresponding object definition screen.



Custom Fields Tab

The following table describes the fields on the **Custom Fields** tab.

Custom Fields Tab

Field	Description
For Category	Select the category to which you would like to add custom fields. All custom fields are category-specific, which means they can be created and viewed by category only. Categories are organized in a hierarchical way in object definitions. Only the top five levels of the hierarchy (inclusive of the first node) can be used when creating custom fields. In system objects, this node is known as Root, while in custom objects this node has the same name as the object itself—such as Dispute for the Dispute object.
Field Name	Enter an alphanumeric name to uniquely identify the field. This name is used to reference the custom field in blocks, letters, XML Layer, and reports. Field names must be unique within each category and use specific naming conventions. See Custom Field Name Requirements. The field name is only editable when a custom field is being defined for the first time. It can never be modified after the field is created.

	Tip: *Try to spell out the entire word rather than making abbreviations so that other solution developers can easily understand exactly what the field is created for.*
Label	Type the field label as you would like it to be displayed on the page. If the field will be included in Data Warehouse, the field label must not exceed 35 characters. Otherwise, the field label can include up to 50 characters. ***Tip:*** *If you need to use different labels for the same field in different blocks (for example, intended for different groups of users), create more custom fields of the type Label Only in the Custom Field Types table.*
Field Type	Select an option from the drop-down list to specify the type of custom field. For more information about each type, see Field Types.
Include in Data Warehouse	Determines whether a field will be processed by Data Warehouse. Select **YES** if you are going to run reports and you want to include the information entered in the custom field for your reports. For information on Data Warehouse, see Data Warehouse Requirements for Custom Fields and Using Data Warehouse.
Is Required?	Select **YES** to make the custom field required for everybody. If you need to make this field required only to certain users, select **NO** and create the appropriate rules. For more information, see Required Fields. By default, required fields are indicated by an asterisk on the page.
Default Value	(Optional) Enter or select a value to be displayed as default for the custom field on the page. For example, a check-box can be displayed as either checked or unchecked.
Exclude from Custom Search	Choose **NO** (the default) or **YES**. If you choose YES, this field will not appear to the end user in custom search screen dropdown lists for criteria or result fields. Specifying Yes can help you simplify searches for the end user by making dropdown lists shorter and less confusing. ***Note:*** *A Yes value causes the field to disappear from the dynamically-generated section of the dropdown list that starts below category names. In the uncommon case that you explicitly define this field as a criterion or result field in*

	a custom search template, then the field will still appear in the dropdown list, in the section above the category names, where all the explicitly defined fields are located.
Custom Field List	Displays the custom fields of the selected object. Select or clear the corresponding check-boxes next to the field and click **edit** or **delete** to change or delete the appropriate custom fields.

1.1.6.3.3 Custom Field Name Requirements

Custom field names must uniquely identify their custom fields and are used in a number of back-end operations to reference those fields.

Aside from making a field name, make sure that each field name meets the following conditions:

- A unique alphanumeric combination
- Unique within its specified category
- Begins with a letter
- Does not include spaces, punctuation, or special characters, that is, a field name must be created from A through Z, a through z, and 0 through 9
- Is no more than 30 characters long, unless it is of the type **Involved**, then it must be no more than 26 characters long
- Is not a reserved word in your database server. (There are dozens of such words; refer to your database documentation for a list.)

1.1.6.3.4 Data Warehouse Requirements for Custom Fields

This section explains Data Warehouse's requirements for custom fields.

The **Include in Data Warehouse** option adds a custom field to a Data Warehouse table for use in reports. All the values of custom fields of a single category that are specified to be included in Data Warehouse are stored in one Data Warehouse table. When you design categories and their reportable fields, you must not exceed Data Warehouse's limits.

Data Warehouse requirements vary depending on whether your system uses an MS SQL Server or an Oracle database. Unless stated otherwise, the following requirements apply to both MS SQL Server and Oracle:

- A category name must be unique within its object if it has any custom fields included in Data Warehouse.

  If a category name is NOT unique within its object, all the corresponding Data Warehouse tables will be created. However, Business Objects create only one class for all the Data Warehouse tables that have the same category name. If the objects within that class mix data from the Data Warehouse tables that have the same category name, the report may include incorrect data.

- Field labels must be unique within a category.

If a field label is NOT unique within its category, the corresponding Data Warehouse table will be created. However, Business Objects will add only one column to the corresponding class in the universe for all the non-unique field labels, and the report may include incorrect information because Business Objects use the data from only one of the fields that has a non-unique field label within one category.

- If a custom field is to be included in Data Warehouse, its field label must NOT exceed 35 characters.

  If a field label exceeds 35 characters, its Data Warehouse table can be created, but Business Objects may not create the necessary object, so reports cannot include the custom field.

  The TeamConnect limit for field labels is 50 characters, regardless of whether the field is included in Data Warehouse.

- For objects of the type **Involved**, the length of custom field names must NOT exceed 26 characters.

  TeamConnect limits the length of custom field names to 30 characters. However, Data Warehouse scripts add a four-character suffix to the custom field names of objects of the type Involved, so the additional four characters could cause field names to exceed the 30-character limit. If the limit is exceeded, creation of the Data Warehouse table fails and an error is logged in the Data Warehouse log file.

- Custom field names must NOT equal any of the words that are reserved for use by your database server. Refer to your database documentation regarding column names for a list of those words.

Oracle Limitations

Data Warehouse tables in Oracle support a maximum of 1000 columns. They do not have a per-row data length limit. That is, in a single category, you can include up to 1000 custom fields in Data Warehouse, regardless of the field type. You must design your categories and reportable fields so that the 1000 column limit is not exceeded.

MS SQL Server Limitations

In MS SQL Server, Data Warehouse tables support a maximum of 8060 bytes per row. You should design your categories and reportable fields so that the 8060 byte limit cannot be exceeded. The following table lists the bytes allocated per column for each field type. These amounts are subtracted from MS SQL Server's maximum allocation of 8060 bytes per row.

For example, if there are 30 custom fields of the type **Text** within a single category, the Data Warehouse table has 30 columns that use 250 bytes each. Altogether, 30 custom fields of the type **Text** use 7500 bytes of the 8060-byte allocation, which only leaves 560 bytes for other field types in that category.

Bytes Allocated for each Field Type (MS SQL Only)

Field type	Bytes allocated
Check-Box	4 bytes

Custom object/ Involved	4 bytes
Date	8 bytes
List type	50 bytes
Memo	16 bytes ***Note:*** *The 16 bytes is for a pointer to the TeamConnect table with the original memo text data, which can contain up to 1 GB in MS SQL Server.*
Number	8 bytes
Text	250 bytes

Important: *When run with MS SQL Server, Data Warehouse has a 250-byte limit for custom fields of the type* ***Text****. If a user enters data in excess of this 250-byte limitation, the Data Warehouse scripts truncate the text when populating the table.*

TeamConnect limits data entry in custom fields of the type **Text** to 2000 characters. However, when the Data Warehouse scripts are run with MS SQL Server, any text that exceeds the 250-byte limitation is truncated.

In addition, some Non-ASCII characters use two bytes per character of the 250-byte data length limitation. To make sure that users do not exceed the 250-byte limit, which results in truncated data in reports, choose one or more of the following options:

- Use memo text fields instead of custom fields of the type **Text** if users might exceed the 250-byte limit.
- Develop rules in TeamConnect to enforce the 250-byte limit for each custom field of the type **Text**.

Data Warehouse tables in MS SQL Server support a maximum of 1024 columns as long as the number of bytes for all fields in the category does not exceed the 8060 byte limit. For example, if a category has 1000 check-boxes and 24 memo text fields included in Data Warehouse, MS SQL Server supports all of the fields because the total allocation is less than 8060:

(1000 * 4 bytes) + (24 * 16 bytes) = 4384 bytes

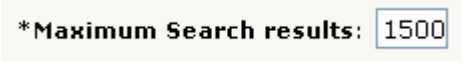
MS SQL Server does not limit the creation of Data Warehouse tables with a large number of columns, although it displays a warning in the Data Warehouse log file about the 8060 byte limit. If a user attempts to insert data into a table in excess of the 8060 byte limit, MS SQL Server displays an error, the system cannot access the Data Warehouse table, and the data cannot be transferred to the Data Warehouse table.

For more information on Data Warehouse, see the Data Warehouse Help.

1.1.6.3.5 Required Fields

A field is considered required if the user cannot save a record without entering a value in that field. If a user attempts to save a record without populating a required field, TeamConnect displays a message that the field is required.

By default, the labels of all required system fields and custom fields (set as required when they were created) appear in boldface with an asterisk next to them, in the end-user interface. However, fields are not automatically displayed this way if they are required as a result of rules, or if another field uses its value. It is easier to require such fields when creating them than to modify the way they appear.



Example: Required Field in End-User Interface

How To Make Fields Required

You may make a field required in one of the following ways:

- By creating a custom field in an object definition and setting it as required for everybody. These fields are automatically displayed as required.
- By creating rules to make system or custom fields required under certain conditions. These fields are not automatically displayed as required.

Fields may also become required (though not marked as such in the user interface) as a result of certain design decisions that you may make. For example:

- If you add an object attribute to the unique identifier (ID) pattern in custom object definitions, the corresponding field becomes required.
- If you make a custom object definition contact-centric, the corresponding contact field becomes required.
- If you set a parent custom object definition as required, the corresponding parent project field in all child projects becomes required.

Practical Tips

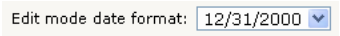
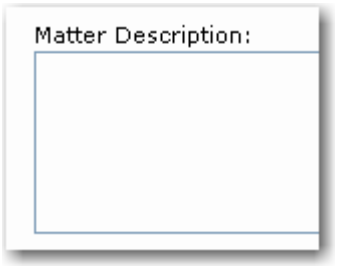
Keep these tips in mind while working with required fields:

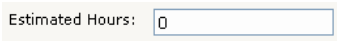
- If you include attributes in a naming pattern for custom object records, consider making the corresponding fields required. This action ensures that these fields are populated and that the resulting descriptions in the **Name** field are more consistent.
- If you make certain fields required in contact records, the users may not be able to add contacts through the contact search module window.
- Use the tag attributes, `notUseRequiredFieldStyle` or `irequired` to control whether custom field labels appear as required when including them in blocks.
- Make sure you include all required fields into wizards that you may be creating for that object. Otherwise, the user is unable to save the record after using the wizard.

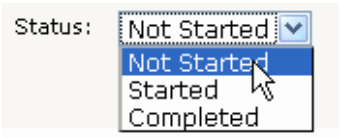
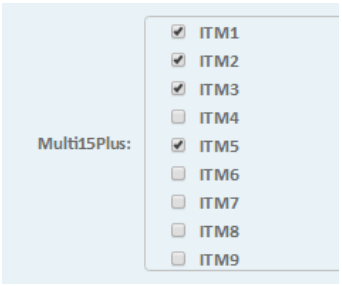
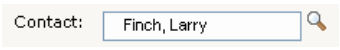
1.1.6.3.6 Field Types

Based on the type of data that each field can store and the field's functionality in the user interface, all system and custom fields in TeamConnect may be one of the seven types listed in the following table.

Custom Field Types

Field type	Description	Resulting field in the end-user interface
Check-Box	Stores boolean values, when you need to provide the user with options that might or might not be selected.	
Date	Stores date values that appear in the format specified in system settings and user preferences. By default, each date field provides the user with a pop-up calendar window that appears when the user clicks in the field. The date and time fields are related. To create a date field with a time field, use the appropriate tag attribute with this field when including it in a form or block file. See Date Field Tag Attributes. When creating a custom field of type Date, a checkbox labeled "Time Zone Independent" appears. You should check this box if the field always contains the same value for any user, viewing it in any time zone in the world. For example, a person's birth date would be time-zone-independent, so the box should be checked. However, the date and time of a telephone conference call would be dependent on time zone, so the box should not be checked. Note that you can only make this choice once, when first creating the custom field. You cannot edit the field later and change this attribute.	
Memo Text	Stores large amounts of text, up to: • About 1 GB of characters (if using ASCII) in MS SQL Server. • 4 GB of character data in the Oracle database. To specify the desired display size for a memo text field, use the appropriate tag attribute with this field when including it in a form or block file. See Memo Text Field Tag Attributes.	
Text	One-line field that can store up to 2000 characters in both MSSQL and Oracle databases.	

	To specify the desired display size for a text field or the maximum number of characters it can accept, use the appropriate tag attribute with this field when including it in a form or block file. See Text Field Tag Attributes. ***Tip:*** *For alphanumeric fields, use a Text field.*	
Number	Stores fixed and floating-point numbers in the following formats: • **Decimal**—The number of digits allowed after the decimal point depends on the decimal format attribute used. Without the attribute, the field accepts as many digits after the decimal point as needed. • **Dollar**—The field accepts a maximum of two digits after the decimal point and automatically displays the dollar ($) sign. • **Percent**—The field accepts a maximum of three digits before the decimal point and as many digits afterwards as needed. It also automatically displays the percent (%) sign. Do not enter any special characters, such as $, %, and so on, as part of default values for this field type. For details on available tag attributes that can be used when including a number field in a form or block file, see Number Field Tag Attributes. ***Tip:*** *For alphanumeric fields, use a Text field.*	
Label Only	The label only field type is available only for custom fields. It allows you to create: • **Several labels for the same field—**Create different labels for the same field and use them in the different blocks. For example, if you need different labels for the same field so that it appears according to different groups of users, you can create a label only field. • **Section headings to help organize a page —**Create sections for groups of custom fields. In either case, you are able to change the section heading or field label in one place, by using a label only field, instead of modifying each individual XML	N/A

	file. It is far more efficient and leaves little room for human error. This field is created like any other custom field. For details, see Creating Custom Fields.	
List	Displays a list of options from which the user can select one, which are provided by a custom lookup table associated with the field. For more information, see List Field Type and Adding Custom Lookup Tables. The example shows a drop-down list. Lists may also be option (radio) buttons.	
Multi-value List	Displays a list of options from which the user can select one or more, which are provided by a custom lookup table associated with the field. For more information, see List Field Type and Adding Custom Lookup Tables.	
Search Module	Allows the user to search and select the desired contact or project record. Search modules allow users to open a selected record directly from the field. There are two types of custom fields that appear as search module fields in the user interface, of type: • **Involved**—Available only in custom object definitions. For details, see Involved Field Type. • **Custom Object**—For details, see Custom Object Field Type.	

1.1.6.3.6.1 List Field Type

Fields of the type List allow the user to select one out of multiple options. In the user interface, they may be displayed as drop-down lists (default display option) or option (radio) buttons.

Important: *To create a field of type List, you must first create a custom lookup table to associate with the field.*

Custom lookup tables are specific to your design and intended to provide lists of options for custom fields of the type List. Typically, each List field has its own custom lookup table that was created for it. For information about custom lookup tables, see Adding Custom Lookup Tables.

Depending on your design, some lookup tables may be re-used in several custom fields—even in different object definitions. For example, a table providing the **Yes/No/Unknown** options for a list of option buttons can be reused for many different fields.

You can change the contents of a custom lookup table by adding, editing, or deleting items to create useful lists that are customized for your design.

Points To Remember about List Fields

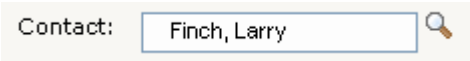
Keep the following points in mind when working with List fields:

- Each List field must be associated with a custom lookup table.
- To create a list of option buttons, create a regular List field and use the `type="radio"` tag attribute when including it in a form or block file. For details, see List Field Tag Attributes.

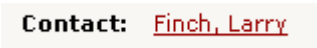
1.1.6.3.6.2 Involved Field Type

Create custom fields of the type Involved to allow users to quickly add involved parties, such as claimant, insured, to a project record. Involved field values can also be used in the auto naming patterns for projects.

In the end-user interface, this field appears as a contact search module field in the edit mode, and as a hyperlink in read-only mode. By clicking the hyperlink, the user is linked directly to the selected contact record.



Contact Field in Edit Mode



Contact Field in Read-Only Mode

Points To Remember

Keep the following points in mind when working with Involved fields:

- Fields of the type Involved can be created only for custom objects.
- To create a custom field of type Involved, you have to select a specific involved role. This is the role of the contact that the user selects in this field.

When you select this field type on the **Custom Fields** tab of an object definition, another drop-down list appears that lists all roles (categories) from the associated Involved object definition.

- When defining an Involved custom field, you can also associate a specific search view with the field. This search view appears when the user opens the search module for this field to select a contact.
- When the user selects a contact in this field in the end-user interface, the following changes occur:
  - On the **Involved** tab of the project, a new Involved record is automatically added for this contact.
  - On the **Involvement** tab of the selected contact record, an entry is added. This tab lists all projects in which the contact is involved.

Search Views for Involved Custom Fields

When defining a custom field of type Involved, you have the option of setting a specific search view to be displayed when the user opens the search module for the field. The search views available for you to select are those search views which are marked as available for use in search modules in the Contact Object Definition, including both **Person and Company Search** and **Company Search Only** search views. For details about these settings, see Defining General Search View Information.

The way that the search module for your custom field of type Involved appears to your users depends on your settings as a solution developer:

- If you want users to always use the same search view in the custom field, then select that search view when defining the custom field.
- If you want users to be able to select which search view to use in the search module for the custom field, you must do the following actions:
  a. Select **None** in the **Search View** field when defining the custom field. This option is selected by default.
  b. Ensure that each contact search view that you want users to have available in search modules is set accordingly (see the General Tab on Search View Screens table).

     The available search views are listed in the user's search module in the **Current View** drop-down list, sorted according to each search view's **Order**.
- If only one contact search view is available for use in search modules, then the available search view is used automatically in the search module. Also, users do not have the option to select a different search view.

This field is created like any other custom field. For details, see Creating Custom Fields.

1.1.6.3.6.3 Custom Object Field Type

Fields of the type Custom Object allow you to establish a non-hierarchical relationship between the current object definition, which can be system or custom, and another custom object definition. Hence, the name of the field type—Custom Object. This field type allows the user to select a related project record directly in the currently displayed record.

In the end-user interface, Custom Object fields appear as project search module fields or drop-down lists in the edit mode. See the Custom Field Types table for examples of these field types.

In the read-only mode, Custom Object fields appear as hyperlinks. By clicking the hyperlink, the user is linked directly to the identified record.

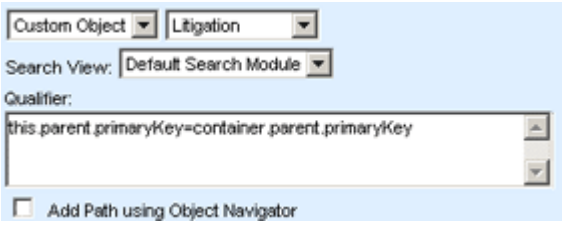
Contact Field with Hyperlink in Read-Only Mode

Points To Remember

Keep the following points in mind when working with Custom Object fields:

- The Custom Object field type is available when creating custom fields for both system and custom object definitions.
- You cannot select system object definitions or embedded custom object definitions when defining fields of the type Custom Object.
- Whether a Custom Object field appears as a search module or a drop-down list in the end-user interface depends on the **Default Selection Mechanism** specified on the **General** tab of the selected custom object definition.

  The display format of the field may also be determined by the display tag attribute in the block file. For details, see Custom Object Field Tag Attributes.
- To limit the available records in a Custom Object field so that a user can select only related records, records in a certain phase, records with a certain default category, and so on, you must add a qualifier in the definition of the field. For more details, see Qualifiers for Custom Object Fields.
- If the field is to be displayed as a search module, you can select a specific search view to be displayed in the search module. Or, you can provide users with the ability to select which search view they want to use in the search module. For more details, see Search Views for Custom Object Fields.
- To specify the desired size for a Custom Object field, you have to use the size tag attribute with this field when including it in a form or block file. For details, see Custom Object Field Tag Attributes.



Data Entry Fields for Custom Fields of the Type Custom Object

Data Entry Fields for Custom fields of the type Custom Object

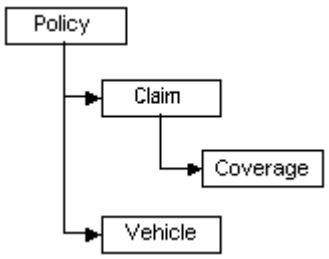
Field	Description

Field Type	Select **Custom Object** to define a custom field of type Custom Object, which allows users to select a project record belonging to the specified custom object definition.
Custom Object List	Select the custom object definition whose records users must be able to select in this custom field.
Search View	This field is only available if the custom object selected in the Custom Object List has search module set as its default selection mechanism on General tab of its object definition. Select the search view that you want users to use in the search module for this field. For details about your other options, see Search Views for Custom Object Fields.
Qualifier	If necessary, define an expression in this field to filter the records available to users. You can define the qualifier by doing one of the following actions: • Type a complete qualifier expression in this field, as described in Qualifiers for Custom Object Fields. • Use Object Navigator to help you create the path on each side of the expression, as described in Creating Qualifiers Using Object Navigator. This field is limited to 256 characters.
Add Path using Object Navigator	Select this check-box to add paths to your qualifier expression with the help of Object Navigator. When this check-box is selected, several additional fields and buttons appear. For details, see Creating Qualifiers Using Object Navigator.

To define a custom field of type Custom Object

1. In the **Field Type** column on the **Custom Fields** tab of the appropriate object definition, select **Custom Object**.

   Several additional fields appear, as shown in the following image.



Object Definition Relationships

2. In the drop-down list which lists custom objects, select the custom object whose records the user must be able to select in the custom field.
3. If you want users to always use the same search view in the custom field, select that search view in the Search View drop-down list.
4. If you want users to be able to select which search view to use in the search module window for the custom field, then you must do the following actions:
   a. Select **None** in the **Search View** field when defining the custom field. This option is selected by default.
   b. Ensure that each search view that you want users to have available in search modules where they select this type of project is set to be available in search modules (see the General Tab on Search View Screens table).

      The available search views are listed in the user's search module in the **Current View** drop-down list, sorted according to each search view's **Order**.
5. If necessary, create a qualifier to filter the available records from which the user can make a selection. You can create the qualifier in one of two ways:
   - In the **Qualifier** field, manually type the appropriate code to create a complete expression. For more details, see Qualifiers for Custom Object Fields.
   - Select the **Add Path using Object Navigator** check-box. This gives you the option to use Object Navigator for each side of the expression. You will have to manually type the operator in the middle of the expression. For more details and instructions, see Creating Qualifiers Using Object Navigator.

For more details on creating custom fields, see Creating Custom Fields.

If the default selection mechanism of the custom object selected in the definition of your custom field of type Custom Object is search module, you can select a specific search view to be displayed to users in the search module.

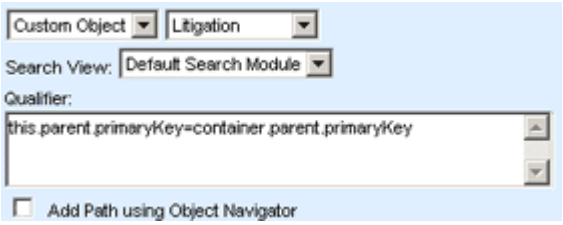
The search views available for you to select are those search views which are marked as available for use in search modules in the Object Definition for the selected custom object. This is an option in the definition of the search view (see Defining General Search View Information).

The search views available in the search module for your custom field of type Custom Object depend on your settings as a solution developer. You can do either of the following operations:

- Select a specific search view that is always displayed in the search module window.
- Allow users to select from multiple search views in the Current View drop-down list, as they typically do in search pages.

If only one search view for the selected custom object is available for use in search modules, then the available search view is used automatically in the search module. Also, users will not have the option to select a different search view.

When creating custom fields of the type Custom Object, you can define qualifiers to filter records of the custom object definition selected for the field. For example, you might want to limit the user's options to records in a certain phase, with parent records of a specific type, and so on. This is done by writing a qualifier expression in the **Qualifier** field on the **Custom Fields** tab of an object definition, as shown in the following image.



Data Entry Fields for Custom Fields of the Type Custom Object

Essentially, in a qualifier expression, you specify a condition to be met in order for records of the selected custom object to be displayed in the custom field. This condition is determined by the business needs of your organization.

Creating a qualifier for a Custom Object custom field is optional; however, it can narrow down the list of records available for selection, which can save users time and prevent invalid selections.

Important: *Creating qualifiers for custom fields of the type Custom Object requires a good understanding of the TeamConnect object model and its attributes. Familiarity with the logic of creating qualifiers in search views and rules is also helpful.*

Points To Remember

The following are the basic points about qualifiers for custom fields of the type Custom Object:

- The code that you provide in the **Qualifier** field must be a complete expression, consisting of two sides that each identify a data value, separated by an operator (such as =). The expression compares two values. For details about the available operators, see Data Types.
- You can create the qualifier either by typing it manually in the **Qualifier** field or with the help of Object Navigator. If you use Object Navigator, you still need to put the various parts together correctly, as described in Creating Qualifiers Using Object Navigator.
- You can use multiple qualifier expressions in the **Qualifier** field by separating them by semicolons (;). They are treated as logical AND statements. In other words, only records that meet the conditions specified by all of the expressions in the **Qualifier** field are returned and available for the user to select.
- When you create qualifiers for custom fields of the type Custom Object in *embedded* object definitions, in the end-user interface, the condition or conditions specified in the qualifier only take effect when the user adds the embedded project and saves its parent first.
- The more complex your qualifier is, the longer it may take for the available records to be displayed when a user wants to make a selection in this custom field.

Whether you decide to create your qualifier by typing it manually or with the help of Object Navigator, you must familiarize yourself with the information in the following sections:

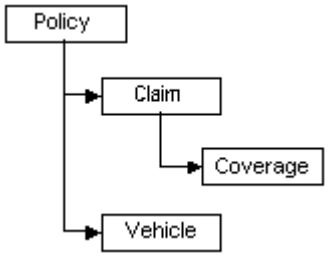
- Qualifier Syntax and Data Types

- Creating Qualifiers Using Object Navigator
- Examples of Qualifiers for Custom Fields of the Type Custom Object

Example

Let's have a look at a specific example of a qualifier in a field of type Custom Object:

Suppose your design consists of several custom object definitions: Policy, Claim, Coverage, and Vehicle. They have the following relationships: Policy is a parent of Claim and Vehicle, Claim is a parent of Coverage (see the following figure).



Object Definition Relationships

In the Coverage object definition, you create a field of type Custom Object and select the Vehicle object in the custom object list (the Data Entry Fields for Custom Field of the Type Custom Object image). In the end-user interface, this means that Coverage records will have a **Vehicle** field where the user selects a Vehicle record.

If you do not write qualifiers for the **Vehicle** field, all Vehicle records in the database are available for selection in this field. You want to limit the choice of Vehicle records only to those records that have the same Policy parent record, as the Claim record, which is the parent of the Coverage record that has the **Vehicle** field.

The following qualifier limits the choice of Vehicle records as desired, by comparing the primary keys of the parent records:

```
this.parent.primaryKey=container.parent.parent.primaryKey
```

In this qualifier expression:

- Keyword this refers to the Vehicle record in the Vehicle field.
- Because Policy is a parent of Vehicle, the parent attribute on the left is the parent Policy record of the Vehicle record.
- Keyword container refers to the Coverage record where the Vehicle field is located.
- Because Claim is the parent of Coverage, and Policy is the parent of Claim, the first parent attribute on the right is the parent Claim record of the Coverage record that contains the field, and the second parent attribute is the parent Policy record of the Claim record.

If the parent record of a Vehicle record is the same as the "grandparent" record of the Coverage record, the Vehicle record is available for selection in the Vehicle field.

There are certain conventions which must be followed in order to construct a valid, complete qualifier expression for a custom field of type Custom Object. In order to create your own qualifier, you must understand both the syntax and the possible data types for an expression.

Syntax

The following points describe the syntax of expressions for qualifiers in custom fields of the type Custom Object:

- The syntax for paths is the same as in the fields where paths are created with Object Navigator, such as rule qualifiers added through the user interface. Namely, the expression you write in the **Qualifier** field essentially consists of object attributes, separated by periods.
- The **Qualifier** field is case-sensitive. Object attributes must be typed exactly as they appear in Object Navigator (see Using Object Navigator) or using the reference tables in Object Model: Read This First plus the tables it points you to. Also, custom field names must be typed exactly as they are defined in the **Custom Fields** tab of the object definition.
- There are two keywords that can be used when writing qualifiers. Always start a path with one of these keywords:
  - `this`

    Refers to the project the user selects in the custom field of type Custom Object. The left side of your expression always starts with this.
  - `container`

    Refers to the record where the custom field of type Custom Object is located. Typically, the right side of your expression starts with container, unless you need to type in a literal value rather than identify an object attribute.
- System fields are identified with the use of a path that identifies the object attribute, like this:

  ```
  this.parent.parent.primaryKey
  ```

  For complete examples of system fields in paths, see the Examples of Qualifiers for Custom Fields of the Type Custom Object table.
- Custom fields are identified with the use of a path that includes the category tree position and custom field name in parentheses, like this:

  ```
  this.detailList(FullTreePosition).detailNumbValueList(FieldName).detailValue
  ```

  where FullTreePosition is the full tree position of the category under which the custom field is created, and FieldName is the name (not label) of the custom field itself.

  Important: *Make sure to type the full category tree position, NOT the name, of the category.*
- When identifying custom field values with a path, you must use the correct path based on the field type, as shown in the Syntax for Identifying Custom Field Values in Paths table. This is because the path changes slightly depending on the type of value you are identifying (number, date, list, and so on), as indicated in **bold** in the table.

  For detailed examples of custom fields in paths, see the Examples of Qualifiers for Custom Fields of the Type Custom Object table.

Syntax for Identifying Custom Field Values in Paths

Custom field type	Path syntax to identify custom field value
Text	`detailList(FullTreePosition).detail`**`Text`**`ValueList(FieldName).detailValue`
Memo Text	`detailList(FullTreePosition).detail`**`Memo`**`ValueList(FieldName).detailValue`
Number	`detailList(FullTreePosition).detail`**`Numb`**`ValueList(FieldName).detailValue`
Date	`detailList(FullTreePosition).detail`**`Date`**`ValueList(FieldName).detailValue`
List (lookup items)	`detailList(FullTreePosition).detail`**`Obje`**`ValueList(FieldName).detailValue`
Custom Object	`detailList(FullTreePosition).detail`**`Objf`**`ValueList(FieldName).detailValue`**`FID`** ***Important:*** *detailValueFID is the primary key of the project record selected in the identified custom field.*
Check-Box	`detailList(FullTreePosition).detail`**`Numb`**`ValueList(FieldName).detailValue` ***Important:*** *Values for custom fields of the type check-box are stored as numbers, where checked=1 and not checked=0.*
Involved	`detailList(FullTreePosition).detail`**`Invl`**`ValueList(FieldName).detailValue.`**`contact.primaryKey`** ***Important:*** *This path identifies the primary key of the contact selected in the Involved custom field.*

Data Types

When constructing an expression for a qualifier, you must ensure that the type of data identified by the path on each side of the expression is the same. For example, you cannot compare a name and a number or a primary key, a string and a decimal, a date and a complex type, or object.

The Data Types and Operators for Custom Fields of the Type Custom Object table lists the types of data that you can use in the **Qualifier** field of a custom field of type Custom Object. It also lists the operators that you can use in the expression for each data type. To find out the data type of a specific attribute, see Object Model: Read This First plus the additional reference tables this reference points you to.

It is important to understand that each side of your expression must identify one of these simple types of data values. Specifically, you cannot compare attributes marked as "bridges" in Object Model: Read This First (and the additional reference tables it points to). These values identify other records, rather than simple data values. For example, parent, createdBy, mainAssignee, and project are all bridges rather than simple data values.

Therefore, when you need to compare records, you must use one of the attributes of the record that is a simple data type and that uniquely identifies it. For example, if you want to compare a parent record to another parent or grandparent record, compare their primary keys. Similarly, if you want to compare a user to another user, compare the username (a text value). For other examples, see the Examples of Qualifiers for Custom Fields of the Type Custom Object table.

Data Types and Operators for Custom fields of the type Custom Object

Data type	Operator	Values	Examples
Text	=	• Path that identifies a text value • A literal text value that you type manually in the **Qualifier** field	`this.parent.name=container.name` `this.currentPhaseType.uniqueCode=LITI`

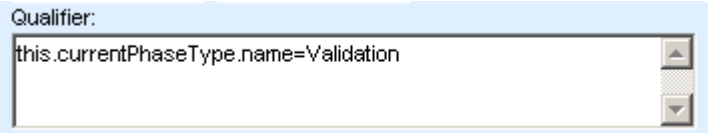
Number	=
	<
	>
	<=
	>=
		• Path that identifies a numeric value
		• A literal numeric value that you type manually in the **Qualifier** field
			`this.parent.primaryKey=container.parent.parent.primaryKey`
			`this.parent.primaryKey=container.detailList(CLAM_AUTO).detailObjfValueList(Vehicle).detailValueFID`
			`this.detailList(RESE).detailNumbValueList(AppraisalAmt).detailValue>=500000`

Date		• Path that identifies a date value
		• TODAY (typed manually in the **Qualifier** field)
		• TODAYX
		For dates, > means after today and <
			`this.createdOn>=container.createdOn`
			`this.closedOn=container.detailList(CLAM_AUTO).detailDateValueList(CompletionDate).detailValue`
			`this.createdOn<=TODAY30`

		means before today.	
Lookup table item (from a system or custom lookup table)	=	• Path that identifies a lookup item selected in a field • A full tree position of a lookup item that you type manually in the **Qualifier** field For an explanation of how tree positions of lookup items are formed for custom lookup tables, see Custom Lookup Tables.	`thisdetailList(CLAM_AUTO).detailObjeValueList(LossCode).detailValue=LOSS_ROOT_VDRE`
Boolean (true/false values)	=	• Path that identifies a boolean value • `true` or `false` typed manually in the **Qualifier** field	`this.isClosed=false`

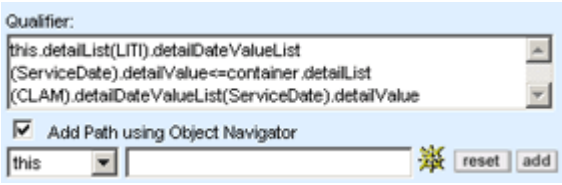
You can use Object Navigator to assist you in constructing the paths on either side of your qualifier expression for a custom field of type Custom Object. This allows you to rely on the internal structure of TeamConnect to guide you in identifying attributes, rather than having to type them.

While Object Navigator helps you construct a path, you must also make the correct selections and type literal text in the **Qualifier** field where appropriate. For example, the following qualifier is constructed using Object Navigator where necessary (currentPhaseType), while a specific value also had to be typed into the field (name.Validation):



Qualifier Example

To take advantage of this option, select the **Add Path using Object Navigator** check-box when defining the custom field, as shown in the following image.



Fields for Adding Paths Using Object Navigator

Even if you decide to use Object Navigator to assist you, you must understand the concepts outlined in the following sections:

- Qualifiers for Custom Object Fields
- Qualifier Syntax and Data Types
- Examples of Qualifiers for Custom Fields of the Type Custom Object

The following table describes the fields for adding paths to define data entry fields for custom fields of the type custom object.

Fields for Adding Paths Using Object Navigator

Field or button	Description
Add Path using Object Navigator	Select this check-box to add paths to your qualifier expression with the help of Object Navigator. When this check-box is selected, several additional fields and buttons appear.
this/container	Select the appropriate keyword to begin the path you are creating: **this**—Select when creating the path on the left side of the expression. **container**—Select when creating the path in the right side of the expression, unless you need to type the literal value you are looking for. These keywords are also described in Syntax.
Object Navigator icon and field	Click the icon to open Object Navigator and build a path. For details on how to use Object Navigator, see Using Object Navigator. Once you have created a path and clicked **ok** in the Object Navigator window, the path appears in the **Object Navigator** field on the **Custom Fields** tab.
reset	Click to clear your selections in the fields specific to creating a path with Object Navigator.
add	Click to add the path you have defined using Object Navigator to the **Qualifier** field. If you have already added any parts of the expression to the **Qualifier** field, the path is added to the end of the string.

Points To Remember

The following points are important to understand about using Object Navigator for defining qualifiers:

- You must begin paths by first selecting either this or container, depending on from which record your path must originate (see the Fields for Adding Paths Using Object Navigator table).
- You may not need to use Object Navigator for both sides of your expression. Often, you need to type the particular text or numeric value you may be looking for, such as a phase name or the tree position of a lookup item, on the right side of the expression.
- If both sides of the expression you need to write require you to build a path, you may need to click the **Object Navigator** icon twice—first to build one path and again to build the other path, one on each side of the operator.
- You must manually type an operator after adding the left side of the expression and before adding the right side of the expression. For details, see Data Types.
- When you build a path and click **add**, the path that you have created is added to the **Qualifier** field.

To build an expression using Object Navigator

1. After specifying the type of the custom field and selecting the desired custom object, select **Add Path using Object Navigator**.

   The screen refreshes to display new fields, as shown in the Fields for Adding Paths Using Object Navigator image.

2. Select **this**.
3. Click the **Object Navigator** icon.

   Object Navigator opens in a new browser window.

4. Navigate to the appropriate object attribute and click **ok**. For details about using Object Navigator, see Using Object Navigator.

   The path you created appears in the Object Navigator field.

5. Click **add**.

   The path, starting with `this.`, is added to the **Qualifier** field.

6. Place your cursor at the end of the path in the **Qualifier** field and type the appropriate operator. For details, see Data Types.
7. If the right side of your expression needs to be a literal value (such as the name of a phase or the tree position of a lookup item), type this literal value after the operator.

   Your qualifier has been defined. You can continue to set the other values to define the custom field, as described in the Custom Fields Tab table.

8. If instead, the right side of your expression needs to identify an object attribute:

   a. Select **container**.

   b. Repeat steps 3 and 4.

c. Click **add**.

Your qualifier has been defined. You can continue to set the other values to define the custom field, as described in the Custom Fields Tab table.

The following examples are typical expressions used in a Qualifier field for a custom field of type Custom Object. You can use these as models for your own qualifiers.

Examples of Qualifiers for Custom fields of the type Custom Object

What you want to do	Qualifier expression	What this expression says
Ensure that projects available for selection in the custom field and the record that contains the custom field have the same parent.	`this.parent.primaryKey=container.parent.primaryKey`	The primary key of the parent of the project record selected in the field equals the primary key of the parent of the record where the field is located.
Filter project records available in the custom field by their phase. Only those records whose current phase is Open can be selected in the field.	`this.currentPhaseType.name=Open`	The name of the current phase of the project record selected in the field equals Open.
In a record where there are two custom fields of the type Custom Object, and the records selected in these fields have a parent-child relationship, filter project records available in one field to only show children of the record selected in the other field.	`this.parent.primaryKey=container.detailList(PAYM).detailObjfValueList(Feature).detailValueFID`	The primary key of the parent record of the project record selected in the field equals the primary key of the project record selected in another field of type Custom Object in the same record.
Ensure that the projects available for selection in the custom field have the same person in a custom field of type Involved as the person identified in the contact-centric field of the record that contains the custom field.	`this.detailList(CLAM).detailInvlValueList(ReportedBy).detailValue.contact.primaryKey=container.contact.primaryKey`	The primary key of the contact who is identified in the ReportedBy custom field in the project record selected in the field equals the primary key of the contact who is selected in the contact-centric field in the record where this field is located.
Ensure that only projects with the specified default category are available for	`this.defaultCategory.treePosition=CLAM_LIAB`	The tree position of the default category of the project record selected in the field is CLAM_LIAB.

selection in the custom field.		
Ensure that only projects with a particular value selected in a custom field of type List are available for selection in the custom field.	`this.detailList(CLAM_AUTO).detailObjeValueList(LossCode).detailValue=LOSS_ROOT_VDRE`	The tree position of the item selected in a custom field in the record selected in the field is LOSS_ROOT_VDRE.
Ensure that projects available for selection in the custom field were created within the past 30 days.	`this.createdOn<=TODAY30`	The createdOn date of the record selected in the field is within 30 days of today or equal to today's date.
Ensure that projects available for selection in the custom field meet three conditions: • The parent of the project is the same as the parent of the record that contains the custom field. • The current phase is Litigation. • The default category is Auto.	`this.parent.primaryKey=container.parent.primaryKey;this.currentPhaseType.name=Litigation;this.defaultCategory.treePosition=CLAM_AUTO`	This qualifier contains three expressions, separated by semicolons (;). See the descriptions for similar expressions in this table.

1.1.6.3.7 Creating Custom Fields

To use blocks for your custom pages, you must first create the necessary custom fields.

To create desired custom fields

1. Analyze your organization's specific information that needs to be captured in the selected object, in addition to the default fields provided within TeamConnect.
2. Decide which custom fields you need to create to capture this information.

   Tip: *Make a list of the labels for these custom fields and their field types.*

3. Make sure the custom lookup tables are created for the custom fields of the type List.
4. Because all custom fields are category-specific, decide to which categories in the object definition you want to add your custom fields.

5. Make sure the desired object categories are added to the object definition.
6. Add your custom fields to the appropriate category in the object definition. See Adding Custom Fields.
7. Include your custom fields into the blocks, depending on your needs and design.

To add a custom field to an object definition

1. In the Designer window, from the **Go to** drop-down list select **Object Definitions**.
2. Select the appropriate object definition from the displayed list.
3. Open the **Custom Fields** tab of the displayed object definition.
4. Select the **Number of entries you would like to add** from the drop-down list.
5. For each data entry row, enter the values in the appropriate fields as described in the Custom Fields Tab table.
6. If you want to continue adding custom fields, click **add more**. Otherwise, click **Save** on the toolbar.

   TeamConnect automatically saves the custom field information and adds the fields to the system block called **Details**.
7. Create the appropriate components for the object view—that is, custom blocks.

Points To Remember

Custom fields can be created and displayed by category only. Because of this category dependency, you must carefully plan for which categories to create the desired custom fields, taking into consideration the following:

- For all categories and custom fields you add in an object definition, associated access rights are automatically created. They must be assigned to the appropriate users and user groups in order for the users to be able to view the categories and their custom fields.
- The root-level category in both custom and system objects is always automatically added to the object records to provide access to other categories (unless you do not intend to use categories and custom fields in your design at all). This means that all custom fields created for this category are automatically displayed in all records for all users.
- Always arrange your custom fields so that you do not have to replicate the same fields for different categories.

  This improves performance and also prevents possible confusion if the custom fields of two categories are used in the same object record.
- A category name must be unique within its object if it has any custom fields included in Data Warehouse. For more information, see Data Warehouse Requirements for Custom Fields.

1.1.6.4 Blocks

Blocks are sets of object information represented by system and custom fields organized in a single XML or JSP file. Blocks typically display as individual sections on the page, as shown in the following image.

System Blocks

System and custom blocks display in the following different ways:

- *System blocks* are included by default in the system object views of TeamConnect. You cannot change, delete, or add new system blocks. For example, on the **General** page of a contact record, **Personal Information**, **Address Information**, and **Phone Numbers** are all system blocks.
- *Custom blocks* are XML files that might refer to Java classes and display sets of system or custom fields organized in sections on record pages or wizard pages.

Custom Blocks

You can use custom blocks to do the following operations:

- Display different system or custom fields for each group of TeamConnect users.
- Design new pages for a record or wizard page.
- Display multiple sets of fields on one record page or wizard page.
- Display custom fields associated with several categories in the same section.

Important: *You must assign the applicable rights for the appropriate categories and details (custom fields) in order for users to be able to view custom blocks that include custom fields.*

Custom blocks in records and wizards are different in the following ways:

Differences between Record and Wizard Custom Blocks

	In records	In wizards

Type of fields you can include in custom blocks	System fields, Custom fields	System fields, Custom fields, Wizard parameters
How users view custom blocks	Add the blocks to object views and assign appropriate rights.	Add the blocks as components to wizard pages.

Button styling

Buttons tags will not have any styling by default. Assign a class (e.g., btn-primary) for styling these objects.

Custom Java Blocks

When you create a screen with custom actions, you may need to create a custom Java block (CJB). The CJB uses the TeamConnect API to specify actions and properties for the custom block. The XML file, which includes the structure, layout, and contents of the custom block, references the Java class. If you are not creating custom actions for the block, you only need to create the XML file without the CJB.

Note: *Do not use a CJB to construct arbitrarily complex JavaScript and HTML code for the custom block.*

1.1.6.4.1 Creating Blocks

When you create a block, you create an XML file using regular HTML tags and formatting in addition to specific TeamConnect tags and attributes that display custom fields, wizard parameters, and their labels on the page. You may also use style sheets, if necessary.

You can edit an XML file for a block manually, using every available TeamConnect block tag. For blocks that are to become part of object views, you can also use the Screen Designer, which has a drag-and-drop GUI that lets you build blocks quickly. The set of block tags available to the Screen Designer is just a subset of all the available block tags in TeamConnect.

See the *Screen Designer Help* for information on creating custom blocks using the Screen Designer tool.

Note: *As a time-saving feature you can start a block's XML file using the Screen Designer, then later edit that file manually to add extra block tags.*

To plan and create a custom block using manual editing

1. Identify the category whose custom fields you want to include in the block.

2. For that category, identify what custom fields each user group has access to.
3. When creating a block for a wizard page, identify the parameters you intend to include in the block and make sure they are defined and added to the appropriate wizard page or pages. For details, see Defining Parameters and Defining Page Components in Wizards.
4. Verify that the desired object category is added to the object definition and the appropriate custom fields are created for the category.
5. Decide how you want to arrange the custom fields (and wizard parameters, if there are any) in the block:
    - Where the fields should be displayed on the page.
    - What the background color of the page should be, if you do not want the system settings or users' preferences to apply.
6. Create the appropriate XML file that contains the necessary fields and structure as explained in Understanding XML Files.
7. Upload the XML file to TeamConnect.
8. Upload any support files, such as scripts, images, or style sheets, to the **Top Level/ HTTPRoot** folder in the Documents area of TeamConnect.
9. Associate the XML file with the appropriate object on the **Blocks** tab of the object as explained in Defining Blocks.
10. If you are creating an object view, include the block in the desired custom object view of the appropriate object. For more information on object views, see Defining Object Views.
11. If you are creating a wizard, add the block to the desired wizard page. For details, see Defining Page Components in Wizards.

Further editing for custom categories

At runtime, a block that was created using Screen Designer may display an error message if all of the following are true:

- The block is part of a custom object definition.
- The category chosen for the block is a custom category.
- The record currently being displayed has not had that custom category assigned to it.

To guard against this possibility, you can do the following:

- Finish laying out the block using Screen Designer.
- Manually edit the resulting XML file, adding a **tc:section** tag that encapsulates the block and specifies the custom category used by the block.

Note: *After this manual edit, the XML file cannot be reopened in Screen Designer, since Screen Designer is unable to parse the tc:section tag.*

Page Element Reference

The table below explains Page Elements, and their properties, in more detail. Where the documentation refers to "expression", the syntax used is Java Expression Language (JEXL) unless otherwise stated.

Page Elements

Element	Property	Notes
Field Column Layout		See the detailed explanation of a layout in step 4.
If Statement		(tc:if) Conditions the display of any elements inside the **if** by evaluating an expression.
	test	The expression to be evaluated.
	negate	If YES, the expression must evaluate false for the contents inside the **if** to be displayed. If NO, the expression must evaluate true for the contents inside the **if** to be displayed. If this property is omitted, it is assumed to be NO.
If-Else Statement		When dragged to the layout, produces two **If** elements, the second of which uses property **negate**=YES. The same value of property **test** should then be entered in each of the two **If** elements.
Java Class		(tc:useClass)
	id	Must be unique within the block. Can be referenced by CSS or Javascript
	name	Required. The name of the Java class. The class file must be located in the same folder that contains the block XML file.
Message		(tc:message) Places text output on the page outside the boundaries of a Field Column layout. Unlike the similar **out** element, a **message** element is localized.
	key	The full i18n key of the message text to be displayed. For information about i18n keys, see Creating Custom Messages.
Message Parameter		This element can only be dragged into an existing **Message** element. Some **Message** elements accept substitution parameters at runtime. For each such parameter in a **Message**, you need a **Message Parameter** element.

	value	Text or expression to be placed in the body of the **Message** element.
Out		(tc:out) Places text output on the page outside the boundaries of a Field Column layout. Unlike the similar **Message** element, an **Out** element cannot be localized.
	value	Text or expression to be rendered in the block.

1.1.6.4.2 Creating Block XML Files

To create an XML file for a custom block, you need to know the following information:

- Basic HTML tags and structure and XML requirements.
- The following custom field and category information if you want to include custom fields in the block:
  - Field name—Field names uniquely identify the fields within a category. You can locate them on the **Custom Fields** tab of the object definition for the block.
  - The full tree position of the category for the custom field—The full tree position of a category is a combination of tree positions assigned to the category and all of its parent categories, starting with the root category and separated by underscores.

    For example, if category B with partial tree position BBBB is created under category A with partial tree position AAAA in the Account object definition, the full tree position of category B is ACCT_AAAA_BBBB.
- Wizard parameter names if you are creating the block for a wizard and want to include parameters. The name of a wizard parameter uniquely identifies the parameter within a wizard. You can locate parameter names on the **Page Components** tab of the wizard page. Define parameters for the page that includes the new block.

  Tip: *To avoid duplicating parameters on the same page, select the* ***Display in blocks only*** *check-box for the parameters that you include in your blocks.*
- The appropriate block tags and formatting.
- The tc:useClass tag with the name of the Java class file and an alias in the `id=` field to refer to the Java class.

Note: *When creating an XML file, you should not hard code background and foreground colors or font type and size. Also minimize the use of JavaScript.*

To create an XML file for a block

1. Open the text editor of your choice.

   Microsoft Notepad is a common choice, but other text editors can be used to create XML files.

2. Type the following lines at the beginning of the document:

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:transform version="1.0" xmlns:tc="http://www.w3.org/1999/XSL/Transform">
```

3. Refer to a Java class using the tc:useClass tag, as shown in the following line:

```
<tc:useClass id="cjb_alias" name="Java_class_name"/>
```

   For `Java_class_name`, enter the name of the Java class. For `cjb_alias`, enter the name of the alias that you use in the XML file to refer to the Java class.

4. Type the contents of your block.

   For a sample of the tags you should include, see Sample XML File.

5. Include the closing tag at the end of the document:

```
</tc:transform>
```

6. Save this new file on your hard drive, with an .xml extension (for example, salesInvolved.xml).

7. Upload the XML file and the Java class to the **Screens** directory of the object for which you are creating the block. See Uploading XML Files.

1.1.6.4.2.1 Block Tags

Custom block files use several TeamConnect tags that allow you to display and organize custom fields and parameters within a block.

TeamConnect Block Tags

Tag	Description
tc:include	Displays a custom block in a custom block.
tc:label	Displays custom field labels in a block.
tc:field	Displays custom fields in a block.
tc:blockTemplate	Displays the title of a block.
tc:wizardParameter	Displays wizard parameter fields in a block created for wizards.
tc:wizardParameterLabel	Displays wizard parameter labels in a block created for wizards.
tc:message	Displays localized messages in a custom block.
tc:messageParam	Displays message parameter values.

For additional block tags that are related to reports, see in Using Tags to Embed Reports in Blocks.

In cases where this topic refers to an "expression" inside a tag, Java Expression Language (JEXL) is the syntax.

Caution: *Do not type field labels or other field information directly in your block files. The formatting specified in the system settings and user preferences do not applied to them. You also have to manually update the labels in all files if the labels change.*

tc:include

This tag displays a custom block within a custom block.

The <tc:include> tag:

- Must always be in lowercase.
- Must use the following attribute:
  - The block attribute that references the block file in the current object definition.
- Uses the following syntax:

```
<tc:include block="BlockName" />
```

  - BlockName—The file name of the block you want to include. If the BlockName attribute does not have a file extension, TeamConnect looks for BlockName.xml, and then for BlockName.scr.xml.

  For example, the following code would include and render a block with file name SampleFile.xml or SampleFile.scr.xml:

```
<tc:include name="SampleFile" />
```

tc:label

This tag displays the label of a custom field. You may use it in blocks created for object views and wizards. Insert tc:label before the tc:field tag to identify the field on the page. For certain field types, such as check-boxes, the label typically displays after the field.

The <tc:label> tag:

- Must always be in lowercase.
- Must use two attributes:
  - The select attribute that references the field name of the appropriate custom field.
  - The category attribute that references the tree position of the field's category.
- Uses the following syntax:

```
<tc:label name="FieldName" category="FullTreePosition" />
```

  - FieldName—The name (not label) of the field you want to include.
  - FullTreePosition—The full tree position of the category for the field.

For example, the following code displays the label "Incident Date & Time" of the custom field with the name "LossDate," created for the root category of the Claim object definition with the unique code CLAM:

```
<tc:label name="LossDate" category="CLAM" />
```

For attributes that you can use with field labels, see Label Tags.

tc:field

This tag displays the field, either blank or with its default value. You may use it in blocks created for object views and wizards.

The <tc:field> tag:

- Must always be in lowercase.
- Must use two attributes:
  - The name attribute that references the field name of the appropriate custom field
  - The category attribute that references the tree position of the field's category.
- Uses the following syntax for system fields:

```
<tc:field name="applicationEntity.FieldName" appropriate attributes/>
```

Uses the following syntax for custom fields:

```
<tc:field name="applicationEntity.FieldName" category="FullTreePosition"
appropriate attributes/>
```

  - FieldName—The name (not label) of the field you want to include.
  - FullTreePosition—The full tree position of the category for the field.
  - appropriate attributes—The appropriate formatting attributes, if necessary.

For example, the following code displays two date fields, ClaimMade and LossDate, both created for the root category of the Claim object definition with the unique code CLAM. The LossDate field, however, appears with a time field next to it because of the showTime formatting attribute.

```
<tc:field name="ClaimMade" category="CLAM" />
<tc:field name="LossDate" category="CLAM" showTime="true"/>
```

For attributes that you can use with tc:field, see Using Tag Attributes.

tc:wizardParameter

This tag displays the parameter field, either blank or with the default value specified for it. You can only use this tag in wizard blocks.

The <tc:wizardParameter> tag:

- Must always use the name attribute that references the parameter name.

- Should not be inside the <tc:section> tag, unless you want to display parameters only if the same section includes the custom fields for the category during record creation.
- Uses the following syntax:

```
<tc:wizardParameter name="ParameterName" appropriate attributes/>
```

  - ParameterName—the name (not label) of the parameter you want to include.
  - appropriate attributes—the appropriate formatting attributes, if necessary.

  For example, the following code displays two number fields, InjuredPersons and ClaimAmount. The ClaimAmount field, however, displays values with the dollar sign ($) before them because of the format attribute used with it:

```
<tc:wizardParameter name="InjuredPersons" />
<tc:wizardParameter name="ClaimAmount" format="DOLLAR"/>
```

  For attributes that you can use with tc:wizardParameter, see Using Tag Attributes.

tc:wizardParameterLabel

This tag displays the label for a parameter field. Insert tc:wizardParameterLabel before the tc:wizardParameter tag to identify the field on the page. For certain field types, such as check-boxes, the label typically displays after the field. If the wizard uses text prompts, you may not need to include the label. You can only use this tag in wizard blocks.

The <tc:wizardParameterLabel> tag:

- Must always use the name attribute that references the parameter name.
- Uses the following syntax:

```
<tc:wizardParameterLabel name="ParameterName" />
```

  - ParameterName—The name (not label!) of the field you want to include.

  For example, the following code displays the label "Method used to report the claim" of the parameter of type List with the name "HowReported":

```
<tc:wizardParameterLabel name="HowReported" />
```

  For attributes that may be used with field labels, see Label Tags.

tc:blockTemplate

This tag displays the title of a custom block. TeamConnect defines the titles of system blocks.

The <tc:blockTemplate> tag:

- Must appear exactly as shown.
- Uses the following syntax:

```
<tc:blockTemplate blockTitle="title_text" />
```

tc:message

This tag allows localized messages to appear in custom blocks.

The <tc:message> tag:

- Uses the following syntax:

```
<tc:message key="messageKey" />
```

  - messageKey—Identification key for the corresponding value.

  For example, the following code displays the value of a common custom key for welcome message text:

```
<tc:message key="custom.common.welcomeMessage" />
```

  Note: *Keys are defined in the Designer on the Custom Messages tab.*

tc:messageParam

This tag, used with the <tc:message> tag, supplies the value for a parameter.

The <tc:messageParam> tag:

- Must be nested inside a <tc:message> tag.
- Replaces parameters in the order of the <tc:messageParam> tag.
- May appear multiple times.
- Uses the following syntax:

```
<tc:message key="messageKey">
       <tc:messageParam value="value" />
   </tc:message>
```

  - value—Argument used to replace the parameter.

  For example, if the defined custom.common value is welcome {0}, the code may appear as follows:

```
<tc:message key="custom.common.welcomeMessageWithUsername">
   <tc:messageParam value="${cjb.currentUsername}" />
</tc:message>
```

Note: *The Designer defines keys on the Custom Messages tab.*

1.1.6.4.2.2 Tags to Embed Reports in Blocks

You can use block tags to embed reports or links to reports within a block.

tc:report

This tag renders the report's chart output within the block. It does not render tabular results.

The <tc:report> tag:

- Must always be in lowercase.
- Must use two attributes:
  - The path attribute (required) that references the location and name of the appropriate report
  - The width and height attributes (optional) that define how much visual space in the block to use for report output. Their default value is 500px.
- Can contain one or more <tc:parameter> tags nested inside the <tc:report> tag.
- Uses the following syntax:

```
<tc:report path="path to report" width="500px" height="500px" />
```

  - path to report—The location and name of the report you want to render. The path is based on the root folder for Reports in TeamConnect. Folders are delimited with the forward slash (/) character.

    Even though each path starts with the name of the root folder, "Reports," the tag also supports omitting the string "Reports" from the beginning of the path and using just a forward slash to imply the root. This shortened form of designating the root folder is valid for all paths in all report tags.
  - 500px—The separate sizes for the width and height of the space for rendering chart output. 500 pixels is the default for both. The suffix "px" or "%" is required at the end of the value to determine the units of measurement. The suffix "px" means that the size is measured in pixels. The suffix "%" means that the size is measured as a percentage of the size of the block itself.

  For example, the following code sample renders chart output for report **OC_Spending**, which is found in subfolder **ManagementReports** beneath the root report folder. The default width and height for the chart output have been overridden with specific values:

```
<tc:report path="/ManagementReports/OC_Spending" width="650px" height="400px"/>
```

tc:reportLink

This tag renders a link within the block. When clicked, the link executes a report. The report appears in a new page, with a link on that page to return to the block that you had been viewing. The user interface of the new page is almost identical to that presented by the **Reports** page when you click on the name of one of the listed reports.

The <tc:reportLink> tag:

- Must always be in mixed case, as shown.

- Uses the following attribute:
  - The path attribute (required) that references the location and name of the appropriate report.
- Can contain one or more <tc:parameter> tags nested inside the <tc:reportLink> tag.
- Uses the following syntax:

```
<tc:reportLink path="path to report"/>
```

  - path to report—The location and name of the report you want to render. The path is based on the root folder for Reports in TeamConnect. Folders are delimited with the forward slash (/) character.

  For example, the following code sample renders a link to report **OC_Spending**, which is found in subfolder **ManagementReports** beneath the root report folder:

```
<tc:reportLink path="/ManagementReports/OC_Spending"/>
```

tc:reportFolderLink

This tag renders a link within the block. When clicked, the link opens a page as if the user had navigated to the **Reports** tab of the application and to the path specified in this tag. The section is equivalent to the section described in Using Report Folders. No reports run as a result of this link, but the user may then select one of the reports within the specified folder.

The <tc:reportFolderLink> tag:

- Must always be in mixed case, as shown.
- Uses the following attribute:

  The path attribute (required) that references the location and name of the report folder.
- Uses the following syntax:

```
<tc:reportFolderLink path="path to folder"/>
```

  - path to folder—The location and name of the report folder you want to display. The path is based on the root folder for Reports in TeamConnect. Folders are delimited with the forward slash (/) character.

  For example, the following code sample renders a link to subfolder **ManagementReports** beneath the root Reports folder:

```
<tc:reportFolderLink path="/ManagementReports"/>
```

  To link to the root Reports folder itself, use either syntax:

```
<tc:reportFolderLink path="/"/>
<tc:reportFolderLink path="Reports"/>
```

tc:parameter

This tag supplies a value to a parameter that a report uses. This tag is only valid within the body of a <tc:report> tag or a <tc:reportLink> tag.

Note: *If a report uses parameters, and a parameter is not specified within the <tc:report> tag or <tc:reportLink> tag, the parameter runs with the default value.*

The <tc:parameter> tag:

- Must always be in lowercase.
- Uses the following attributes:
  - The name attribute (required) corresponds to the parameter name that the report designer defines. If the name value is not resolved at report runtime, an error message appears.
  - The value attribute (required) specifies the value associated with this parameter during the execution of the report.

    The value attribute can contain a list of multiple values, separated by semicolons.

    If the value(s) for this attribute are not resolved at report runtime, an error message appears.

    Permissible values are dependent on the datatype of the parameter. Most datatypes allow the value of the parameter to be expressed as a string literal, as long as that string conforms to the datatype. For example, a date datatype accepts the string "2011-04-30" and a category datatype accepts the string "EXPE_ABCD". Datatypes that do not permit string literals include **User**, **Project**, **Contact**, **Involved**, and **Account**. For those five datatypes, you must use an attribute path expression.

    Parameters of datatype Enum accept a string literal composed of one or more Java constant names. Multiple names are delimited by semicolons (;). For more information about Enum (enumerations) see Enumerations.

    All datatypes, whether or not they permit string literals, accept an attribute path expression that references an object compatible with the datatype. Examples of attribute path expressions are ${cjb.startedOn} for a date parameter and ${applicationEntity.treePosition} for a lookup table parameter.
- Uses the following syntax (shown within a containing tc:report tag):

```
<tc:report path="path to report" width="500px" height="500px">
<tc:parameter name="name" value="value"/>
<tc:parameter name="name" value="value"/>
</tc:report>
```

  - path to report—The location and name of the report you want to render. The path is based on the root folder for Reports in TeamConnect. Folders are delimited with the forward slash (/) character.
  - 500px—The separate sizes for the width and height of the space for rendering chart output. 500 pixels is the default for both. The suffix "px" or "%" is required at the end of the value to determine the units of measurement. The suffix "px" means that the size is measured in pixels. The suffix "%" means that the size is measured as a percentage of the size of the block itself.
  - name—The name of the parameter that the report designer knows.

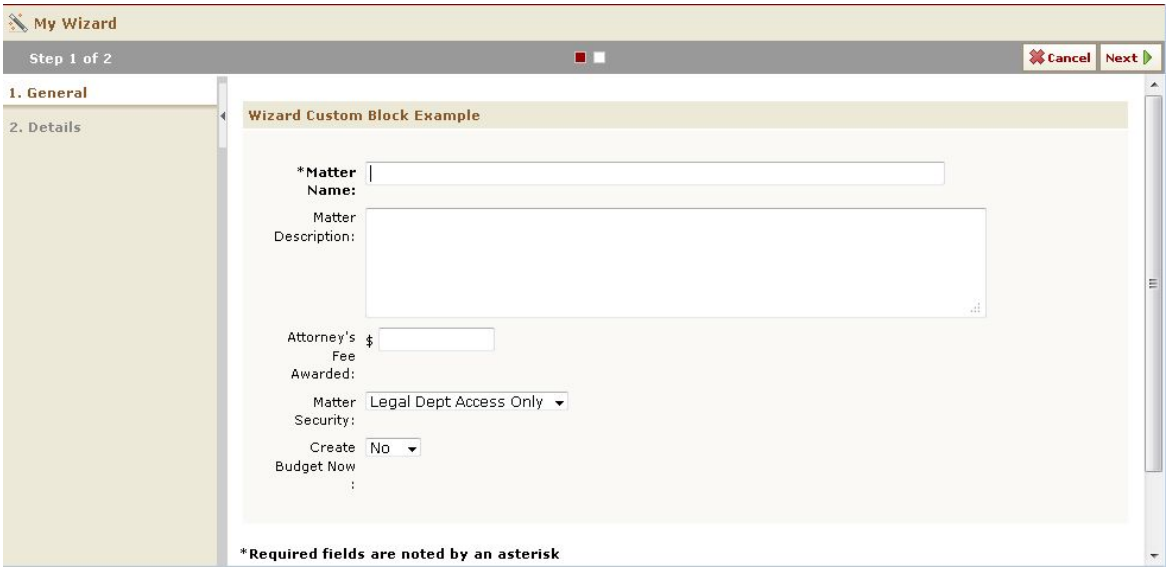
- value—The object or literal that contains a value to be associated with the parameter for this execution of the report.

For example, the following code sample renders chart output for report **RejectedInvoices**, which is found in subfolder **ManagementReports** beneath the root report folder. The default width and height for the chart output are overridden with specific values. A report parameter named "pastXdays" is assigned a value contained in the attribute path "limitDays":

```
<tc:report path="ManagementReports/RejectedInvoices" width="650px"
height="400px">
    <tc:parameter name="pastXdays" value="${cjb.limitDays}"/>
</tc:report>
```

1.1.6.4.2.3 XML File Sample

This topic contains an XML code sample that results in the **General Subpoena** block for the following wizard page.



Note: *You can customize the tabbing order by adding a tabIndex attribute to each of the <tc:field> and <tc:wizardParameter> tags within an XML file. For details, see tabIndex.*

The XML sample in this topic demonstrates how to include the following examples in an XML file:

- Using the `<tc:useClass>` tag to refer to the `MyCustomBlock.class` file.
- Using the `<tc:blockTemplate>` tag with the optional `blockTitle` attribute.
- Adding custom fields to a block. This custom field in this code sample are text, number, and lookup table fields.
- Using formatting tag attributes for the number field, such as `format` and `size`.
- Adding wizard parameters to a block. Because this XML sample contains parameters, you can only use it in wizards. To use the code sample in an object view, remove or comment out the parameters from the code.

- Specifying a custom JavaScript file to add styles to the CJB.

The custom block image shows the resulting block on a wizard page.

Code Sample

```
<?xml version="1.0" ?>
<tc:transform version="4.0" xmlns:tc="http://www.mitratech.com/schemas/2008/
custom">
   <tc:section category="DISP">
      <tc:blockTemplate blockTitle="Wizard Custom Block Example">
      <!-- Use a custom java class -->
      <tc:useClass id="cjb" name="MyCustomBlock"/>
         <div class="bg_blockcolor">
            <table cellspacing="10" width="100%">
               <tr>
                  <td width="50%" align="left" valign="top">
                     <table cellpadding="3" width="96%">
                        <!-- System field -->
                        <tr valign="top">
                           <td align="right" class="tdlabel2">
                              <tc:label colon="true" required="true"
                              key="custom.common.legal.matterName"></
                              tc:label>
                           </td>
                           <td>
                              <!-- Default value of field generated by
                              custom java class -->
                              <tc:field maxLength="250"
                              name="enterpriseEntity.name" size="77"
                              value="${cjb.generateName}"></tc:field>
                           </td>
                        </tr>
                        <!-- Custom field of type text -->
                        <tr valign="top">
                           <td align="right" class="tdlabel2">
                              <tc:label colon="true"
                              name="MatterDescriptionDI" category="DISP"></
                              tc:label>
                           </td>
                           <td>
                              <tc:note cols="80" category="DISP"
                              name="MatterDescriptionDI" rows="5"></tc:note>
                           </td>
                        </tr>
                        <!-- Custom field of type number -->
```

```
                        <tr valign="top">
                            <td align="right" class="tdlabel2">
                                <tc:label colon="true"
                                name="AttorneysFeeAwardedDI"
                                category="DISP"></tc:label>
                            </td>
                            <td>
                                <!-- Number value formatted as currency -->
                                <tc:number name="AttorneysFeeAwardedDI"
                                category="DISP" format="CURRENCY" size="12"></
                                tc:number>
                            </td>
                        </tr>
                        <!-- Custom field of type lookup -->
                        <tr valign="top">
                            <td align="right" class="tdlabel2">
                                <tc:label colon="true" name="MatterSecurityDI"
                                category="DISP"></tc:label>
                            </td>
                            <td>
                                <tc:lookup nullValueDisplayString="(None
                                Selected)" category="DISP"
                                name="MatterSecurityDI" tableCode="SECE"
                                allowNullValue="true"></tc:lookup>
                            </td>
                        </tr>
                        <!-- Wizard parameter -->
                        <tr valign="top">
                            <td align="right" class="tdlabel2">
                                <tc:wizardParameterLabel
                                name="CreateBudgetNow" colon="true"></
                                tc:wizardParameterLabel>
                            </td>
                            <td>
                                <tc:wizardParameter name="CreateBudgetNow"
                                wizardUniqueKey="DNDI"></tc:wizardParameter>
                            </td>
                        </tr>
                    </table>
                </td>
            </tr>
        </table>
    </div>
<!-- Include custom javascript uploaded to Top Level > System > HTTPRoot
-->
<script src="./httproot/myjs.js" type="text/javascript"> </script>
```

```
        </tc:blockTemplate>
    </tc:section>
</tc:transform>
```

1.1.6.4.3 Uploading Custom Block Files

After you have created the required XML file and Java class you must upload them to the **Screens** directory of TeamConnect.

To upload CJB files to TeamConnect

1. Log in to TeamConnect and Designer, then in the **Go To** drop-down list select **Documents**.
2. Browse to the **Screens** directory of the required custom or system object for which you are creating the block.

   For example, if the required object was named Claim, you would browse to **Top Level \System\Object Definitions\Claim\Screens**.
3. Click **Upload Document** on the toolbar.
4. From the **Upload New File** screen, choose one of the files to upload. Enter the **File Name** and **File Type** and click **upload file**.

   You can do the same for any other files.

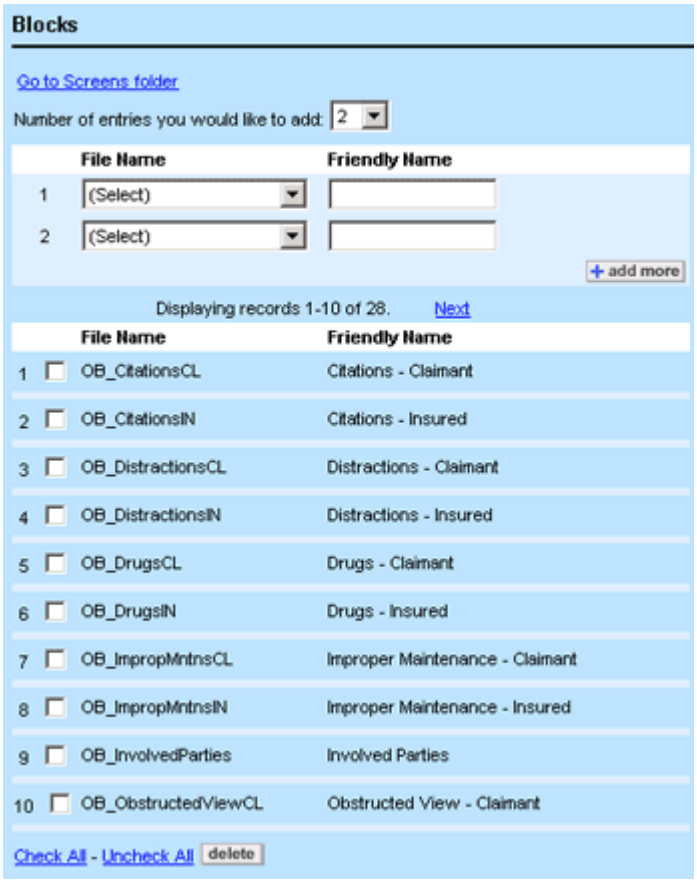
The name of the file now appears on the **File Name** drop-down list on the **Blocks** tab of the system or custom object. To define the block in TeamConnect, see Defining Blocks.

1.1.6.4.4 Defining Blocks

After you have written the XML file that contains the structure and content of the desired block, you need to define it as a block for TeamConnect. You can do this by selecting the XML file on the **Blocks** tab of the appropriate object definition.

To add a block to an object definition

1. In the Designer window, from the **Go to** drop-down list select **Object Definitions**.
2. Select the appropriate object definition and then select the **Blocks** tab.



Custom Blocks Tab

3. Enter the appropriate values as described in the Blocks Tab table.
4. If you want to continue adding custom blocks, click **add more**. Otherwise, click **Save** on the toolbar.

The blocks are now available for inclusion in custom object views or wizards for the selected object definition. To view a block on the screen, you must include it in an object view (for details, see Creating Object Views), or if it contains parameters, you must add it as a page component in a wizard (for details, see Defining Page Components in Wizards).

The following table describes the items in the **Blocks** tab.

Blocks Tab

Item	Description
Go to Screens folder	Click this hyperlink to go to the **Screens** directory of the selected object. This directory is where you store all your XML files (blocks) for the selected object.
File Name	Select the XML file that contains the block information.

	Note: *This drop-down list reflects the contents of the Screens directory of the selected object.*
Friendly Name	Enter a name (or title) to identify the block. This name is used to reference the block when adding it to an object view or a wizard page, and it is best to reflect the contents of the blocks. The maximum length is 250 alphanumeric characters. ***Tip:*** *Make the friendly name as descriptive as possible so other solution developers can easily understand what the block is created for.*
Block List	Displays the blocks for the selected category. Select or clear the corresponding check-boxes on the left to change or delete the appropriate blocks.

Friendly Names

Friendly names are distinguished by objects, so you cannot use the same name twice for the same object, but you can use the same name for other objects. For example, you can use the friendly name "Assignees" in all objects, except for custom objects and the Task object because they already use the Assignees name.

The following table includes the words that you cannot use as friendly names for custom blocks.

Friendly Names You Cannot Use

Objects	Words You Cannot Use
All Objects	• General • Categories • Detail Forms • Documents • Security • Access Info • History • Group Rights • Workflow
Accounts	• Amount • Posting Criteria • Posting Criteria Task

	• Posting Criteria Expense • Posting Criteria Involved • Posting Criteria Vendor • Posting Criteria Invoice • Transactions • Deposit/Withdraw • Transfer • Child Accounts
Appointments	• Attendees • Resources
Contacts	• Address • Phone • Email • Fax • Internet Address • Skills • Rates • Employee • Relations • Territories • Involved
Documents	• Version
Invoices	• Line Items • Transaction
Involved Parties	• Account • Relations
Projects	• Phase Histories • Assignees

	• Relations • Milestone • Task • Appointment • Expense • Account • Involved
Tasks	• Assignees • Billing Information • Transaction

1.1.6.4.5 Troubleshooting Custom Blocks

Custom Block Troubleshooting Tips

Possible problem example	Possible solutions
The time fields do not appear in date fields.	Make sure you are using the correct formatting tags and set their values properly. See Using Tag Attributes.
Numbers do not display with correct currency symbols, percentages, or decimal points.	
Option buttons and search modules appear as drop-down lists.	
Fields are too wide or too narrow.	
Fields and labels do not appear on the page.	Check the following: • The correct spelling of custom field and parameter names. • The full category tree positions. • Correctly created sections for each category. • The correct spelling of field and label tags. • The rights to all categories and their custom fields.

	• The categories, with fields you do not see, added to the record you are testing.
The layout of the entire block does not display properly.	Make sure the `colspan` for each table row is set correctly if the tables use the `colspan` attribute.
Nested tables do not display properly. The background color is broken.	Make sure each nested table is enclosed between the `<tr><td>` and `</tr></td>` tags in the XML file.
The system displays a message about parsing errors.	Troubleshoot the XML file. See Understanding XML Files.
The system displays a message about block errors and refers to an error log.	Check the error log for more information.
The block does not reflect changes to an XML file.	Make sure you have checked out and checked in the XML file properly in the **Documents** area.
The uploaded XML file fails to appear on the page.	Check for the following: • The uploaded file defined as a custom block in the object definition. • The custom block part of a custom view that is assigned to the whole system or to the default group in your user account. • The block added as a page component to the wizard. • The **.xml** extension of the file in lowercase.
Required custom fields do not display in red (or another specified color).	Check for the following: • In the XML file, fields use the `tc:label` tag and parameters use the `tc:wizardParameterLabel` tag. • A different color is not hard coded in the XML file.

1.1.6.5 Object Views

Object views are pages created to display object records. These views consist of sets of tabs providing complete information about a record. Within TeamConnect we distinguish between two types of object views, system and custom.

The Designer interface uses "tab" to signify a page in the end-user interface. However, switching between pages in that interface is accomplished by clicking on links in the left pane, not by clicking on tabs. In general in this section, a tab corresponds to a left-pane link.

System object views are the predefined record pages created for TeamConnect objects (both system and custom). You can see what each system view includes, but you cannot change, delete or add new system views.

Custom object views are the pages that you, as the TeamConnect solution developer, create for your organization to meet its specific needs or individual preferences. In other words, you can create your own custom views, change, and delete the existing custom views, as necessary.

If different groups are going to see different object views, assign each view to the appropriate groups with their **Group Account** pages.

If the rest of the TeamConnect users who do not have a default group set in their user accounts are going to see the same custom view, assign it globally in the **System Settings** screen.

By default, all object views that appear to the TeamConnect users are set on the **Default Object Views** tab of the **System Settings** screen, where all objects have their system views automatically selected as their default views. To change these settings, you have to first create your desired custom views for the necessary objects and then assign them as default one for the whole system.

Note: *If you decide to keep the system object views, you do not have to create or define object views. If you decide to create and define custom object views, continue through the sections that follow.*

For all objects, you have the option of either using the system views or creating your own custom views as follows:

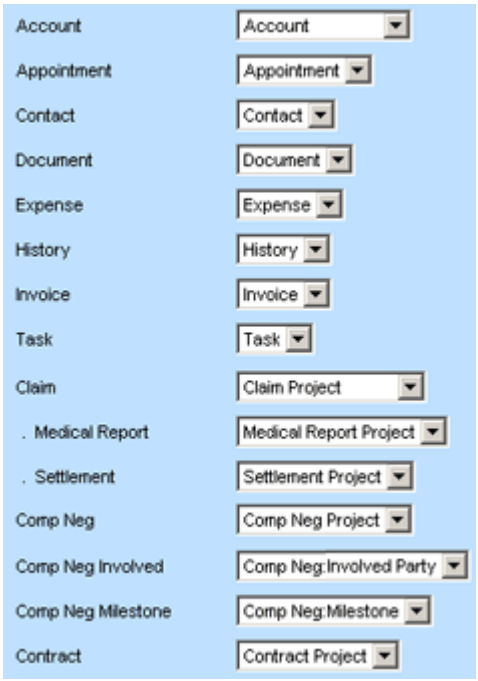
- Defining Object Views
- Creating Object Views
- Assigning Object Views

Object View Names

There are certain object view naming conventions that are followed when system views are created for all objects. This makes it easier for you to distinguish between the system and custom views of the system and custom objects.

- All *system views* of the *system objects* have the same name as the objects themselves, for example account, contact, and so on.
- All *system views* of the *custom objects* have the name of the object with the word "Project" after it, for example Claim Project, Policy Project, Litigation Project, and so on.
- All *custom views* of *all objects* have the names you, as the solution developer, assigned to them when creating them. See Creating Object Views.

The following image shows a Designer example of system object views as they appear on the **Default Object Views** tab in the **System Settings** screen.

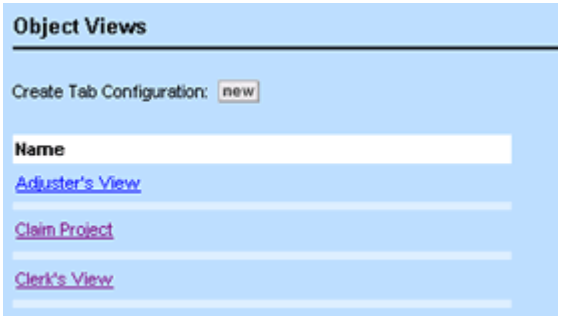


Default Object Views Tab on System Settings Screen

To open an existing object view

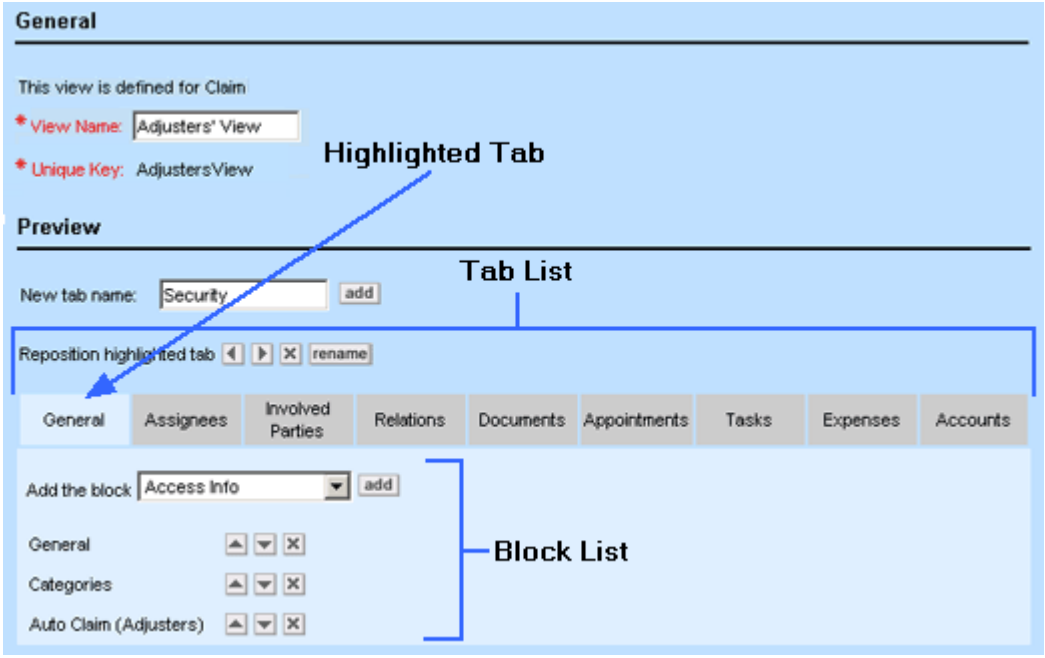
1. In the Designer window, from the **Go to** drop-down list, select **Object Definitions**.
2. Select the appropriate object definition and then select the **Object Views** tab.

   The corresponding **Object Views** screen appears.



Object Views Tab (Adjuster's View is the Custom View and Clerk's View is the System View)

3. Click the required object view to open its Setup screen.



Object View Screen

The **Object Views** screen is divided into two sections:

- **General**, which displays the name of the view, the object to which the view is assigned, and the object view's unique key.
- **Preview**, which displays the tabs in the object view and the order of the tabs, and allows you to add blocks to the highlighted tab.

Object View Screen

Field or control	Description
View Name	Enter a name to identify the object view. This name should specify the group for which this view is intended. Maximum 250 characters long. ***Note:*** *When first naming a view, you must* ***Save*** *the view to display other sections of the screen.*
Unique Key	Enter an alphanumeric combination to identify the object view within the object definition. The unique key must be 50 or fewer characters in length, cannot contain spaces, underscores, commas, punctuation marks, or any special characters. Once you save the object view the first time, you cannot change its unique key.

		Object views within the same object definition cannot have the same unique key, but object views belonging to different object definitions can have the same key.
New tab name		Enter the name of a new page you would like to add to the object view. Click **add** to add the page.
Reposition highlighted tab		Click to reposition the selected tab to the left or right of its current position. ***Note:*** *Click a tab to select it. The selected tab does not have a border color; tabs that are not selected have a gray border around them.*
		Click to delete the selected tab, then click **OK** in the confirmation dialog box.
	rename	Click, enter a name in the pop-up window to rename the selected tab, and then click **OK**.
Tab List		Lists all tabs present in the object view, and their current order. ***Note:*** *Click a tab to select it. The selected tab does not have a border color; tabs that are not selected have a gray border around them.*
Add the block		Select a block to add. ***Note:*** *The options in this drop-down list include all system blocks and all custom blocks defined for the selected object.*
Block List		Lists all blocks present on the selected tab, in the current order. ***Note:*** *Click the **up** and **down** arrows next to the block name to reposition the block above or below its current position, or click **delete** next to the block name.*

1.1.6.5.1 Defining Object Views

Defining custom object views is the last step in the process of customizing what information different users or groups see in TeamConnect.

Custom object views typically consist of the system blocks and custom blocks. Before creating and assigning object views, you should consider the following questions:

- Have you defined all categories necessary for the object?

  All custom fields and blocks are category-specific, which means they can be created and displayed by category only. For information on defining categories, see Using Categories.

- Have you created all custom fields necessary for the object?

  Custom fields are the central component of object views, as they are used to store information specific to your organization, and are used in blocks. For information on defining custom fields, see Creating Custom Fields.

- If you are going to use custom blocks, have you created all custom blocks you need for the object view?

  Blocks are sets of information that are included into different object views created for different groups of users. For information on defining blocks, see the Screen Designer Help.

If you answered yes to all questions that apply to your needs, you are ready to create your own custom object views. After you have defined your object views, you must assign them to the appropriate groups or the entire system.

1.1.6.5.1.1 Creating Object Views

To create an object view, you need to name the view, add all necessary tabs to it, and lay out the blocks on each tab in the proper order.

Tip: *Make a list of tabs you would like to include in the object view you are creating and a list of blocks that you want to include on each tab.*

To create a custom object view

1. Open the **Object Views** tab of the appropriate object definition.
2. Click **New**.

   You can also create a copy of an existing object view by opening the appropriate object view of the displayed object definition, and then clicking Create a Copy.

   The **General** section of the new object view appears.
3. In the **View Name** field, type a name for the new object view.
4. In the **Unique Key** field, type a name or value that will uniquely identify the view within the object definition.

   An object view's unique key cannot contain spaces, underscores, punctuation marks, or any special characters.
5. Click **Save**.

   The **Preview** section of the new object view appears.
6. On the **New tab name** field, type the name the tab that you would like to create in your custom object view, and click **add**.

7. In the **Add the block** drop-down list, select a block that you would like to add to the selected tab, and click **add**.

   The selected block appears under the selected tab in the **Preview** section.

8. Repeat step 7 for as many blocks as you would like to add to the selected tab.
9. Position the blocks you have added in the desired order by clicking the up and down arrows ( ) next to each block.
10. Repeat steps 6 and 9 for all tabs you would like to add to your custom object view.
11. Position the tabs you have added in the desired order by selecting the appropriate tab and clicking the right and left **Reposition highlighted tab** arrows ( ).
12. Click **Save**.

    To see your new object views in the displayed list, click **Refresh**.

    To allow users to view and use your object views you must assign them to the designated groups or set the view as default in the system settings. See Assigning Object Views.

Points To Remember

Consider the following when creating custom object views:

- The more blocks you add to a tab, the longer it takes to open it.
- Each tab that you create appears in the end-user interface only when at least one block is added to the tab.
- Blocks for related objects are not displayed in the record until the user saves it first.
- The **General** block must be always added to the custom object view, as it typically contains the required system fields. Make sure to include all required fields in every custom object view.
- Because all custom components of an object view are category-specific, make sure you add the **Categories** block to the object view. Otherwise, the users are unable to add categories and consequently see their custom fields.

  Task and Expense do not have the **Categories** block. Instead, there is a field called **Default Category** in their **General** block.

  The Contact object presents its category-related custom field values in the **Details** page, not in the **General** page.

- If you want child or embedded object records to be displayed in the selected object's records, you must add the corresponding blocks to a tab in the object view and name it accordingly.

  These blocks are automatically created when the child or embedded custom objects are created.

  For example, if you want each parent Policy object record to have its child Claim records and its embedded object Coverage records displayed on the same tab, you must create this tab (for example, called Claims and Coverages) in the custom object view in the Policy object definition and add the blocks called Claim and Coverage to that tab.

Important: *If you want system object records, such as appointments, accounts, tasks, and expenses, to be displayed in a project, you must create tabs or sections with the appropriate names and add the corresponding system blocks.*

1.1.6.5.1.2 Assigning Object Views

By default, all users see the system views for all objects as specified on the **Default Object Views** tab of the **System Settings** screen. After a custom object view is created, you must assign it either to a group or to the entire system before users are able to use that object view.

The object views that are set in the **System Settings** screen are going to be displayed for a user who:

- Does not belong to a user group
- Has no default group selected in his or her user account
- Has the **(Select a View)** option displayed next to the custom object in default **Object Views** tab of his or her default group.

Note: *The default object views set in group accounts always take precedence over the system settings.*

To assign object views to the system:

1. In the Designer window bar, in the **System Settings** drop-down list select **Default Object Views**.
2. Scroll to the object to which you want to assign a view and do one of the following actions:
   - If you want to set a certain custom view created for the object as default for the system, select the name of that view in the drop-down list next to the object.
   - If you want to set the system view of the object as default for the system, select that view in the drop-down list next to the object. For details on how to identify a system view in the list, see Object View Names.
3. Repeat step 2 for all objects to which you want to change default views.
4. Click **Save**.

   The selected settings take effect immediately, and the users who do not have a default group set in their user accounts see the object views assigned to the system.

1.1.6.5.2 Troubleshooting Object Views

The following table presents common object view problems and their solutions.

Custom Object View Troubleshooting Tips

Possible problems	Possible solutions

The custom view that I created is not displayed.	Check the following information: • The view is assigned either to the whole system or to the default group in your user account. • If you have assigned the view to a group, make sure the user account you are using for testing is a member of the group, and the group is set as default in this account.
My custom blocks are not displayed in my object view.	Check the following information: • Make sure the blocks are defined and added to the tabs in your view. • Make sure that the **Categories** block is added to the view. • Troubleshoot your custom blocks. See Troubleshooting Custom Blocks. • Clear the cache. See Clearing Custom Blocks Cache. See also Points To Remember.
Not all of my tabs appear on the page.	Make sure that each missing tab has at least one block added to it.
The blocks for related objects are not displayed in the record.	Save the record first and then check again.

1.1.6.5.3 Clearing Custom Blocks Cache

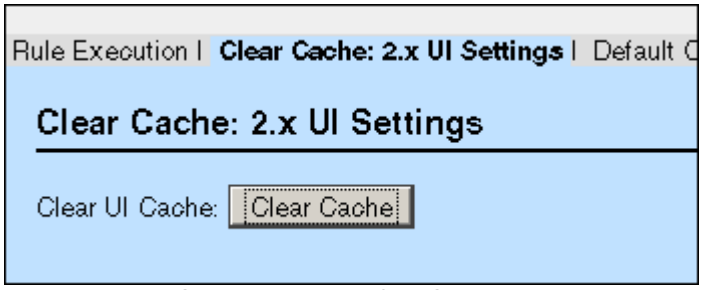
TeamConnect has a user interface caching mechanism that stores page designs and other user interface data accessed by the application. You should clear the cache whenever you make changes to blocks and object views, and after making other changes to the **Main** user interface.

Important: *If you make a change that requires clearing the cache but you do not clear the cache at that time, users do not see the change.*

To clear the custom blocks cache

1. In the Designer window, from the **System Settings** drop-down list, select **Clear Cache 2.x UI Settings**.
2. On the displayed tab, click the **Clear Cache** button.

The cache has been cleared and you can close the **System Settings** screen and continue with your work.



System User Interface Settings

1.1.6.5.4 Customizing Invoice Line Item Views

For invoice records, there is an additional capability of defining two views for invoice line items. Unlike other custom views, this capability is limited to a number of system fields that will be displayed in a predetermined layout, which means that you do not have to create any XML files or define custom object views. All you can do is to select the fields for each view, specify their default values, if any, and allow their values to be copied to new line items.

Note: *If your organization has a custom tool for creating additional custom fields for invoice line items, you may have additional capabilities that are out of scope of this documentation.*

You can customize the appearance of the **Line Items** block which can be placed in any Invoice Object View's tab. Users can also customize the location of the Line Item Block inside the Invoice Object View through the end-user interface in the Invoice Object Definition View. Here, users can create a new tab for the line item block or place the line item block in the **General** tab.

You can customize the fields that appear in the **Line Items** block on the user interface. Users have the ability to set field defaults and can make the view as similar to their business practice as necessary.

The following table describes the items in the **Line Items** customization tab.

Line Items Customization Tab

Field or column	Description
Field	The name of the field that appears on the **Line Items** tab in an invoice.
Required	Specifies whether the field is required. Select the check-box for the **Activity** and **Timekeeper** fields to make these fields required for Tasks. Important: If you have upgraded from a previous version of TeamConnect, leave the **Timekeeper** check-box selected to ensure backward compatibility with existing invoice records.
Line Item Entry	For each field, select whether to show it or hide it in the **Line Item Entry** screen.

Table Display	For each field, select whether to show it or hide it in the **Table Display** screen.
Task Adjust	For each field, select whether to show it or hide it in the **Batch Adjust** screen. For information on hiding certain fields in the In-Line & Bulk Adjustment screen, see Customizing Invoice Line Item Views for In-Line & Bulk Adjustment.
Expense Adjust	For each field, select whether to show it or hide it in the **Batch Adjust** screen. For information on hiding certain fields in the In-Line & Bulk Adjustment screen, see Customizing Invoice Line Item Views for In-Line & Bulk Adjustment.
Default Value	Select a default value for each field.
Copy Capability	For each field, select the check-box to indicate that the value of the field is copied from the previous line item to the new line item when a user clicks the **Copy** icon in the **Action** column.

Customizing Invoice Line Item Views for In-Line & Bulk Adjustment

The behavior of line item views is different for the In-Line & Bulk Adjustment screen.

In the In-Line & Bulk Adjustment screen, the **Date** field will not honor the Line Items Customization settings and will always display.

Additionally, the **Type**, **Category**, and **Timekeeper** fields will display in the In-Line & Bulk Adjustment screen unless both **Task Adjust** and **Expense Adjust** are set to **Hide** for each respective field.

Search Views

There are two search views of the **Line Items** block that you can use to view line item details—*All Line Items* and *Line Item Warnings*. Users have the ability to toggle between these two views, set one as default, and can make the two views as similar or different as necessary. For more information on line item views, see the *User Guide*.

1.1.7 Creating Search Views

To help users find records, you can create search views that define different combinations of search criteria fields and how the results appear.

Users search for records by entering search criteria in the resulting search screens. Lists of matching records appear with various columns of information to help users recognize and access the desired record. For example, on the **Tasks** search screen, users may select various search views from links in the left navigation pane to help them find their own pending tasks or others' overdue tasks.

TeamConnect contains many predefined search views for objects installed by default. It also allows you to customize the search views for each object to include a variety of fields, providing users with choices of how they want to search for and display records.

Uses for Search Views

A custom-defined search view can be designated as a collection. When you create such a collection, the collection name appears as a link in the left navigation pane of the end-user interface. Clicking on that link runs the search and displays the search results in the work area of the user interface. In this way a search with complex filtering criteria can be reused quickly, saving the end user the trouble of entering those criteria each time. You can also define, and then assign search views to groups of users.

You may use search views not only for finding records, but also to create a summary of or report on records, which you may print directly from the search screen. For example, you may use a search view for creating a report of all expenses posted during the last 30 days.

When you define a search view, you must also specify where it may be used or accessed in any of the following ways: as a related object, through a portal pane, as a collection link for the object, and through a search module. This flexibility allows you to design different search views to meet the needs of users depending on where they need to search for records.

Searching with Related Objects

When a search view is used with an object that may have other kinds of objects related to it, the search results are limited to records that actually do have related objects present. For example, this combination might happen if a custom object has a category associated with it, which specifies a relation with Involved objects.

If a specific record of that custom object type has the specified category assigned to it, but does not have any relationships with Involved records currently, that record is omitted from the search results.

1.1.7.1 Designing Search Views

Creating search views is one of the last steps in the process of defining objects in TeamConnect as described in Customization Sequence.

The procedure for creating a search view is described in detail in Creating Search Views.

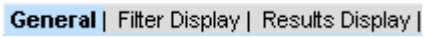
When you are ready to create search views for an object, first determine the following characteristics for each search view you want to create:

- Where the search view should be available, for example, from multiple user-interface locations or a separate search view for each location. The available locations are:
  - Related objects
  - Portal panes
  - Search modules
  - Links in the left pane (for search views that are designated as collections)

In addition, one search view can be set to display Global Search results for the object.

- Whether to display search results automatically when the user selects the search view (Auto Search).
- Which fields are available as search criteria.
- Which qualifiers are invisible, so that records are filtered by certain invisible criteria in addition to the criteria entered by the user.
- Which section titles are needed for the **Filter** tab (the search criteria section).
- Which fields appear on the **Results** tab (the displayed records section for records found by the search) in addition to the object link.
- If the search view is for a contact or appointment, whether you want to use the predefined Card or Calendar display types.

All search views are created within the Designer object definition area for their objects in TeamConnect. Search views may be created, viewed, and modified in the **Search Views** screen, which contains the following tabs:



Search Screen Tabs

- **General**—Contains general information about the search view, such as the name, view type, locations in TeamConnect where it is available, and whether it is set to auto search. For more details, see Defining General Search View Information.
- **Filter Display**—Contains the qualifiers defined for the search view with which users can conduct a search. This tab also includes invisible qualifiers that automatically limit all search results, to help you control which records users can access. For more details, see Search View Filter Display.
- **Results Display—**Contains the settings that determine how the search results appear for the user in the displayed records section or the **Results** tab of the search screen. For more details, see Search View Results Display.

1.1.7.1.1 Accessing the Search View Screen

To open a search view screen

1. Open the appropriate object definition, and then click the **Search Views** tab.

   A list of all existing search views for the object appears, as shown in the following image. *This example is on the Contact Object*.

Search Views

Create Search View: new

	View Name	Order	Used For
1	AAContact Search	10	Search Module(Person and company search)
2	ALL Contact all fields	10	
3	All Contacts	1	Object Collection
4	Company Search Module	1	Search Module(Company only search)
5	custom fields hover over	10	
6	custom field sub category	10	Custom Search
7	Custom Search	0	Custom Search
8	Default Search Module	0	Search Module(Person and company search)
9	Demo Search view	10	
10	Global Search Default	10	
11	Global Search View	10	Global Search
12	komal hover over	10	
13	Law Firms	4	Related Object, Object Collection
14	Legal Department Staff	5	Related Object, Object Collection
15	Outside Counsel Attorneys	6	Related Object, Object Collection
16	Recently Created	2	Object Collection
17	Recently Modified	3	Object Collection
18	Search for Companies	0	Portal Pane, Search Module(Company only search)
19	Search for Contacts	0	Related Object, Search Module(Person and company search)
20	Search for Court	13	Portal Pane, Related Object, Search Module(Company only search)
21	Search for Judge	10	Portal Pane, Related Object, Search Module(Person and company search)
22	Search for Outside Counsel Attorney	11	Portal Pane, Related Object, Search Module(Person and company search)
23	Search for Staff	12	Portal Pane, Related Object, Search Module(Person and company search)
24	Search for Vendor	14	Portal Pane, Related Object, Search Module(Company only search)
25	subcategory hover over	10	Custom Search
26	system fields hover over	10	Custom Search

Check All - Uncheck All delete

Use this search view's results display for global search: Global Search View

Use this search view's results display for contact groups and recently viewed: All Contacts

Use this search view for workflow approvals: (None Selected)

Use this search view for workflow processes: (None Selected)

Use this search view for hover over display popups: Demo Search view

Sample Search Views Tab Screen

2. Click the hyperlink of the search view you to want to open, or click **new** to create a new search view for the selected object definition.

   The corresponding search view screen appears with its **General** tab (see the General Tab on Search View Screens image) displayed by default.

3. Click the tab you need to open.

The following table describes the items in the **Search Views** tab.

Search Views Tab Screen Elements

Field or column	Description
Create Search View:	Click **new** to create a new search view. This displays a new **Search View** screen.
View Name	Displays the name of each search view as a hyperlink. Click the name of the search view to open it.
Order	Displays the display order in which all search views for this object appear in the **Current View** drop-down list on the search screen. The first search view in the list is the search view that appears by default when a user first accesses a search screen. For details about setting the display order, see the General Tab on Search View Screens table.
Used For	Displays all areas where the search view is available, such as search modules, custom search, and portal panes. Not all objects allow search views to be used in all areas. For details about where search views can be used, see the General Tab on Search View Screens table.
Check All - Uncheck All	To delete a search view, select a single check-box next to the search view or click **Check All** to select all the search views on the page, and click **delete**. Click **Uncheck All** to deselect all the search views.
Use this search view's results display for search across objects	Select a search view from the drop-down list to display Global Search results. For system objects, the default search view is **Global Search Default**. For custom objects, TeamConnect uses the search view with the lowest display order. If more than one search view shares the same sort order, TeamConnect uses the first search view in alphabetical order.
Use this search view's results display for static object collections and recently viewed	Select a search view from the drop-down list as the default to display static object collections. (See **Object Collection** in the General Tab on Search View Screens table) This search view's results display is also the display used for Recently Viewed records for this object. Recently Viewed is one of

	the available collections in the left navigation pane of the user interface. *When working with search views for the Contact object, the label for this check-box will instead say "Use this search view's results for contact groups (Address Books) and recently viewed." A Contact Group is a form of static object collection.*
Use this search view for workflow approvals	This field is visible for all objects that can have approval rules. Select a search view from the drop-down list to display on the **Requests to Approve** page of **My Approvals** and the **Active Requests** page of **My Requests**. The fields under the **Filter Display** tab of the search view display as the fields you use to enter search criteria. The fields under **Results Display** of the search view display as the columns in the table with the search results.
Use this search view for workflow processes	This field is visible for all objects that can have approval rules. Select a search view from the drop-down list to display on the **Attention Required** and **In Progress** pages of **My Workflow Processes**. The fields under **Filter Display** of the search view display as the fields you use to enter search criteria. The fields under **Results Display** of the search view display as the columns in the table with the search results.
Use this search view for hover over display popups	Select a search view here to display when items within the object are hovered over with the mouse. For more information on this field and setting up custom hover pop-ups, please see Custom Hover Popups.

1.1.7.1.2 Creating Search Views

The following instructions for creating a search view provide a general overview of the entire process. This process includes several general steps that are broken down into greater detail in other instructions, as indicated in the appropriate step.

To create a search view

1. Open the appropriate object definition, and then click the **Search Views** tab.
2. Click **New**.

   The **General** tab appears.

3. Enter all general information about the search view on the **General** tab.

   For details, see Defining General Search View Information.

4. Click the **Results Display** tab and do the following actions:

   a. If you want users to be able to access records directly from the search results, create the **Object Link**.

      For details, see Defining Object Links.

   b. Design the other display columns for the results screen of the search view.

      For details, see Creating Results Display Columns.

5. Click **Save** to save the search view. This allows you to add qualifiers.

6. Click the **Filter Display** tab and do the following actions:

   a. Create all desired section titles for the search view.

      For details, see Creating Section Titles.

   b. Create all desired search qualifiers for the search view.

      For details, see Creating Search View Qualifiers.

7. Click **Save**.

   The search view is now available in the appropriate screens in the **Current View** drop-down list. After you save a search view, you may edit it as necessary.

1.1.7.1.2.1 Creating Search Views for Home Pages

You can use a search view that you have created to display useful information to users in their TeamConnect **Home** page, such as the tasks to which they are assigned, their daily appointments, and projects for which they are the main assignee.

Search views that are created for use in home pages have several requirements:

- The **Portal Pane** check-box on the search views **General** tab must be selected. If this check-box is not selected, then the search view is not available in the **Content** section of the **Portal Pane** screen.
- Qualifiers with predefined values are required in order for the search view to display properly in the portal pane. If no search qualifiers are defined on the **Filter Display** tab of the search view, then the portal pane lists all available records in the database for that object. If desired, the qualifiers can be invisible.

  For example, you might want the portal pane to list the current user's tasks to which they are assigned, rather than the tasks for all users.

- You should also take into consideration which extra qualifiers you want to have available when the user clicks **Edit Filter** in the portal pane, and make these qualifiers visible qualifiers.

  For example, you might want to give the user the ability to further limit the Tasks that appear according to the date which they are due.

- You must ensure that each search view has an object link display key in order for users to have direct access to records from the search view.

  Card and **Calendar** display types for the Contact and Appointment objects are not available for search views in home pages.

The following steps describe how to create a search view that you can use in a portal pane. To learn more about using search views in portal panes, see Adding Search Views.

To create a search view that can be used for home pages

1. Open the appropriate object definition, and then click the **Search Views** tab.
2. Click new.

   The **General** tab appears.
3. On the **General** tab, do the following actions:
   a. In the **Used For** list of check-boxes, select **Portal Pane**.
   b. Enter all other general information about the search view on the **General** tab.

      For details, see Defining General Search View Information.
4. Click the **Results Display** tab and do the following actions:
   a. Create the **Object Link**.

      This is a necessary element for search views in portal panes in order for users to be able to access records displayed in the portal pane. For details, see Defining Object Links.

      Do not confuse this with the **Object Link** type of portal pane content.
   b. Design the other display columns for the results screen of the search view.

      For details, see Creating Results Display Columns.
5. Click **Save** to save the search view. This allows you to define qualifiers.
6. Click the **Filter Display** tab and do the following actions:
   a. **Create all desired section titles for the search view. The section titles appear when the user clicks Edit Filter in the Home page.**

      For details, see Creating Section Titles.
   b. Create all desired invisible qualifiers with predefined qualifier values for the search view. Invisible qualifiers automatically filter the records that appear on the user's Home page in the portal pane.

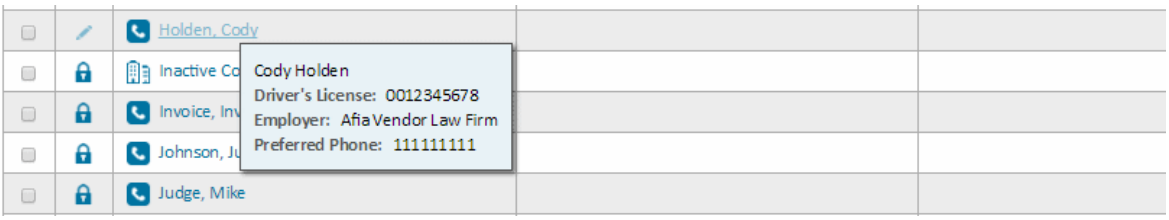
      For details, see Visible and Invisible Qualifiers.
   c. Create all desired visible qualifiers for the search view. Visible qualifiers are available when the user clicks **Edit Filter** in the Portal Pane.

      For details, see Visible and Invisible Qualifiers.
7. Click **Save**.

The search view that you have created for the current object definition is now available for use when creating portal panes. For more details, see Adding Search View Object Links.

1.1.7.1.2.2 Custom Hover Popups

TeamConnect designers are able to create custom pop-up views for items on an object when hovered over. When creating these custom views, it is important to note that the hover is built entirely from the **Results Display** tab. Please see the walkthrough below for further detail.

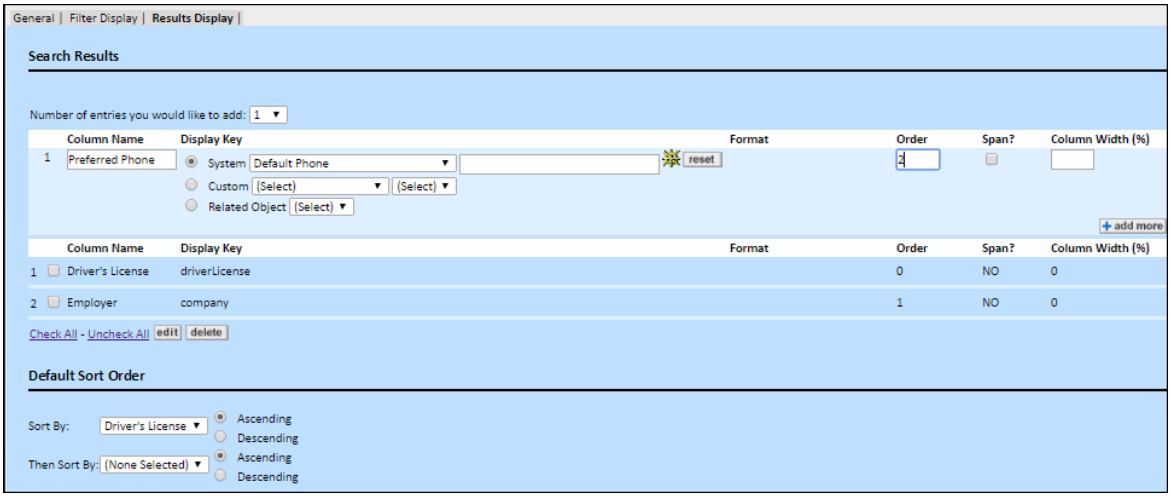


Customized Hover Pop-Up

Creating a Custom Hover Over

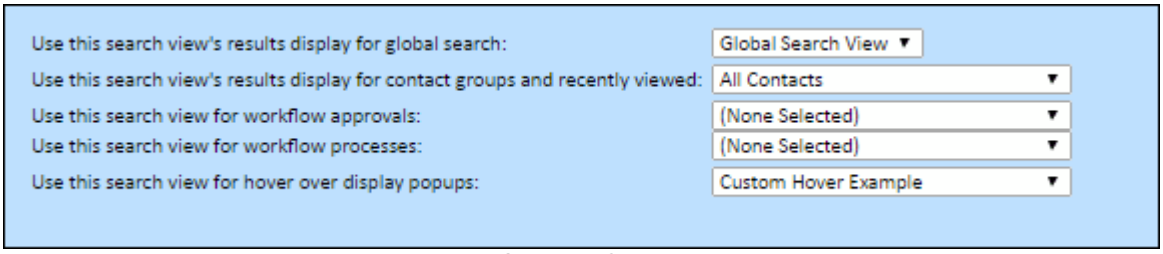
Note: *Existing search views can be used for custom hovers.*

1. Click the **Setup** link in the upper-right corner of TeamConnect.
2. From the **Go to...** drop-down list, select **Object Definitions.**
3. Click the desired object for the custom hover. The custom hover can be applied to any system or custom object. Select the **Search Views** tab.
4. Create a new Search View.
5. From the General tab, ensure that none of the checkboxes under **Used for:** are selected. Enter in the desired name and unique key.
6. Click the **Results Display** tab. Add the desired system or custom fields to be viewed by entering column name, selecting the appropriate item from the dropdown, and adjusting the order as desired. More details can be found on Search View Results Display. Further sorting details can be adjusted from the **Sort by:** dropdowns at the bottom of the screen.



Custom Hover Results Display

7. After saving, navigate to the **Search Views** page. Select the created view for the **Use this search view for hover over display popups.**



Applying Selected Custom Hover

1.1.7.1.2.3 Specifying Search Views Used by Global Search

The *Global Search* feature in TeamConnect's user interface allows users to search for specific object types, **All** object types or across **All Projects**. Global Search searches all of the applicable fields for the user's search string, so you do not have to configure any search criteria for Global Search.

However, Global Search uses the **Results Display** configured for each object's search view.

When searching across **All** or **All Projects**, Global Search uses the search view specified in the following drop-down list on the object's **Search Views** tab to display its results:

Use this search view's results display for search across objects

For system objects, the default global search view is **Global Search Default**. For custom objects, Global Search uses the search view with the lowest display order. If more than one search view shares the same sort order, Global Search uses the first search view in alphabetical order.

Global Search does not use the search view's **Filter Display**. Instead, it searches all of the objects' fields that are supported by Global Search. Memo text fields are also searched For full-text searching, see Full-Text Search Requirements.

To specify a search view for Global Search

1. Open the appropriate object definition, and then click the **Search Views** tab.

   A list of all existing search views for the object appears.

Search Views

Create Search View: new

	View Name	Order	Used For
1	AAContact Search	10	Search Module(Person and company search)
2	ALL Contact all fields	10	
3	All Contacts	1	Object Collection
4	Company Search Module	1	Search Module(Company only search)
5	custom fields hover over	10	
6	custom field sub category	10	Custom Search
7	Custom Search	0	Custom Search
8	Default Search Module	0	Search Module(Person and company search)
9	Demo Search view	10	
10	Global Search Default	10	
11	Global Search View	10	Global Search
12	komal hover over	10	
13	Law Firms	4	Related Object, Object Collection
14	Legal Department Staff	5	Related Object, Object Collection
15	Outside Counsel Attorneys	6	Related Object, Object Collection
16	Recently Created	2	Object Collection
17	Recently Modified	3	Object Collection
18	Search for Companies	0	Portal Pane, Search Module(Company only search)
19	Search for Contacts	0	Related Object, Search Module(Person and company search)
20	Search for Court	13	Portal Pane, Related Object, Search Module(Company only search)
21	Search for Judge	10	Portal Pane, Related Object, Search Module(Person and company search)
22	Search for Outside Counsel Attorney	11	Portal Pane, Related Object, Search Module(Person and company search)
23	Search for Staff	12	Portal Pane, Related Object, Search Module(Person and company search)
24	Search for Vendor	14	Portal Pane, Related Object, Search Module(Company only search)
25	subcategory hover over	10	Custom Search
26	system fields hover over	10	Custom Search

Check All - Uncheck All delete

Use this search view's results display for global search: Global Search View

Use this search view's results display for contact groups and recently viewed: All Contacts

Use this search view for workflow approvals: (None Selected)

Use this search view for workflow processes: (None Selected)

Use this search view for hover over display popups: Demo Search view

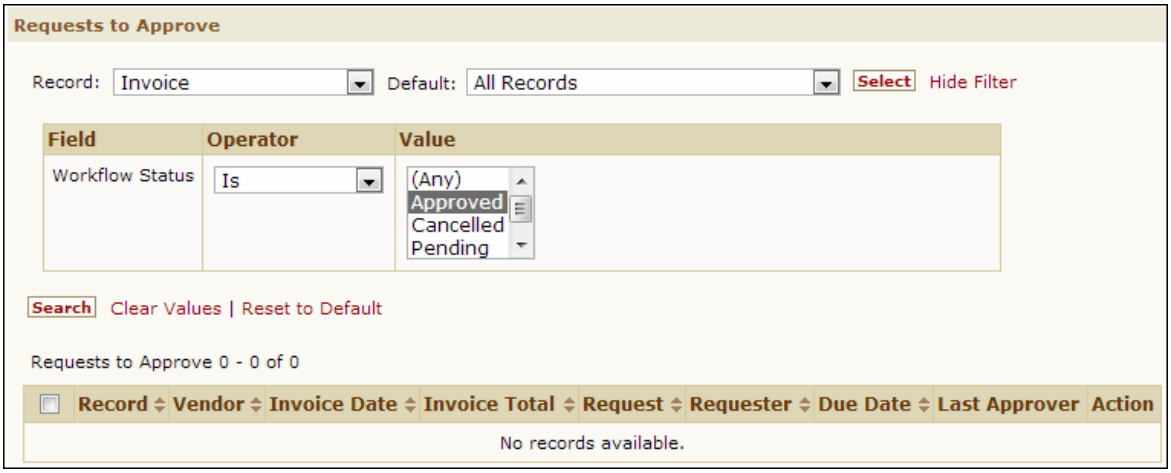
Sample Search Views Tab Screen

2. Select the desired search view from the **Use this search view's results display for search across objects** drop-down list.

You can use a search view that you have created to add a search filter to the following pages in TeamConnect:

- **Requests to Approve** of **My Approvals**
- **Active Requests** of **My Requests**
- **Attention Required** of **My Workflow Processes**
- **In Progress** of **My Workflow Processes**

When you select a search view for these pages, the fields under the **Filter Display** tab of the search view display as the fields you use to enter search criteria. The fields under the **Results Display** tab of the search view display as the columns in the table with the search results.



Requests to Approve Page with a Search Filter

To create search views for Approvals, Requests, and Workflow Process Pages

1. Open the appropriate object definition.
2. Click the **Search Views** tab.

   A list of all existing search views for the object appears (see the Sample Search Views Tab Screen image).
3. Depending of the pages where you want to add the search view, select a search view from one of the following drop-down lists:

Page in TeamConnect	Search Views Drop-Down List
• **Requests to Approve of My Approvals** • **Active Requests of My Requests**	Use this search view for workflow approvals
• **Attention Required of My Workflow Processes** • **In Progress of My Workflow Processes**	Use this search view for workflow processes

1.1.7.1.2.4 Search View Requirements for Full-Text Searching

Full-text searching allow users to find records even if they lack the name or number of a record. For example, the name of a city might be mentioned in the notes of the record. To find the record, the user would search by the content of the corresponding field or use Global Search.

To find a document that has been uploaded to the Documents area, users can search for specific text strings, such as a person's name that is mentioned in the document.

To allow users to search the contents of the following areas of TeamConnect, you must add the corresponding search qualifiers to your search views:

- **Notes** system field in system object records
- Custom fields of type Memo Text
- **Description** system field in History records
- Content of text-based documents that are uploaded to TeamConnect, such as:
  - Microsoft Word®, Excel®, PowerPoint®
  - Novell/Corel WordPerfect®
  - Adobe Acrobat® (PDF)
  - ASCII text files
  - HTML files
  - Other file types supported for text queries by Oracle Text or SQL Server Full-Text. For a detailed list, see your database server documentation.

Note: *Global Search does not support document searches. For more details, see Specifying Search Views Used by Global Search.*

Full-Text Search Requirements

To provide content searching capability to users, the following requirements must be met:

- You must configure and enable Oracle Text (Oracle interMedia) or SQL Server Full-Text Search on your TeamConnect installation. This ordinarily happens by default during installation.
- You must add the corresponding fields in search views as qualifiers, as explained in Adding Document Content Searching to Search Views and Adding Field Content Searching to Search Views.

If you want to give users the ability to perform document content searching, you must provide a search view with this feature.

By default, a search view for the Document object definition, called **Content Search**, is provided with TeamConnect. However, you can also add the capability of searching by document content to your own custom search views, using the system field called **Document Body** as a search qualifier.

To add content search qualifiers to a document search view

1. Open the **Document** object definition and click the **Search Views** tab.
2. Do one of the following actions:
   - To change an existing search view, click the name of the search view to open it.
   - To create a search view, click **new** and do the following actions:
     i. Complete the **General** tab.

     For details, see Defining General Search View Information.

     ii. On the **Results Display** tab, add at least one column to the results display.

     This allows you to save the search view. For details, see Search View Results Display.

     iii. Click **Save**.

     After the search view is saved, you can define search qualifiers.
3. Click the **Filter Display** tab and create a search qualifier using the system field **Document Body**.

   This field appears on the **Filter** tab or the search criteria section of the object's search screen so that users can enter the text content that they are seeking. For details, see Creating Search View Qualifiers.
4. If you are using Oracle as your database server and want the **Score** field to be displayed on the **Results** tab or the displayed records section of the object's search screen, click the **Results Display** tab and select the system field **Document Body**.
5. Click **Save**.

   The content search qualifier is now available in the search view.

To allow users to search according to the field contents, you can specify the following fields as search qualifiers on the Filter Display tab of a search view:

- **Description** field in history records
- **Notes** field in system records
- Custom fields of type Memo Text

Content search fields will be displayed in the search criteria section of the search screen so that users can type the text that they are seeking. A question mark (?) hyperlink displayed next to such fields allows users to select search operators.

You can also add the content search fields on the **Results Display** tab so that users can see the entire contents of the fields in the results. For details about field types and their storage sizes, see Field Types.

To add a search qualifier for record content searching

1. Open the appropriate object definition and click the **Search Views** tab.

2. Do one of the following actions:
   - To change an existing search view, click its name to open it.
   - To create a search view, click **new** and do the following actions:
     i. Complete the **General** tab.
     ii. On the **Results Display** tab, add at least one column to the results display.
     iii. Click **Save**.

     Once you save the search view, you can define its search qualifiers.
3. Click the **Filter Display** tab and create a search qualifier using the following:
   - **Notes** system field in a system object
   - **Description** system field in the History object
   - Custom fields of the type Memo Text in a system or custom object

   To verify that a custom field is a Memo Text field, locate it on the **Custom Fields** tab of the object definition and check its field type.
4. (Optional) Click the **Results Display** tab and add the field that you selected as the search qualifier.
5. Click **Save**.

   The content search qualifier is now available in the search view.

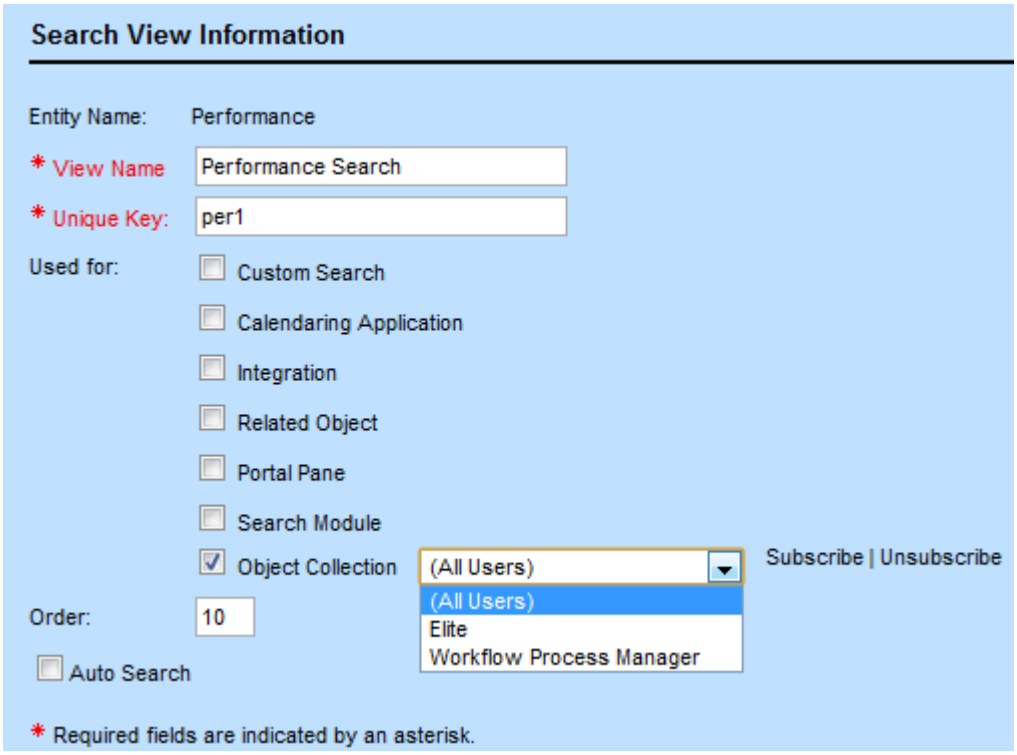
1.1.7.1.3 Defining General Search View Information

General search view information includes the name of the search view, the object for which it is defined, and other general details. These settings are found on the **General** tab of search view screens and determine which search view for an object appears by default in various screens.

To define general search view information

1. Open the appropriate object definition, and then click the **Search Views** tab.
2. Select the appropriate search view, or click **new** to create a search view.

   The **General** tab of the search view appears.



General Tab on Search View Screens

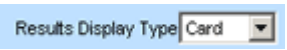
3. Complete the fields in the **General** tab as described in the General Tab on Search View Screens table.

   You cannot save the search view until you create a results display, unless **Card** or **Calendar** is selected in the **Results Display Type** drop-down list for a contact or appointment search view.

Once you specify the general information for the search view, you can create the results display. See Search View Results Display.

General Tab on Search View Screens

Field	Description
Entity Name	Automatically displays the name of the object for which this search view is defined.
View Name	Enter a descriptive name for the search view as you want it to appear in the left pane of the corresponding search screen (maximum 250 characters). ***Note:*** *It is recommended not to create a name that includes special characters. For example: ?, \, /, |, :, <, >, *, %, %, #, &, (,), {, }, [, , _, ". If the view is enabled for IMAP synchronization (IMAP Search), then special*

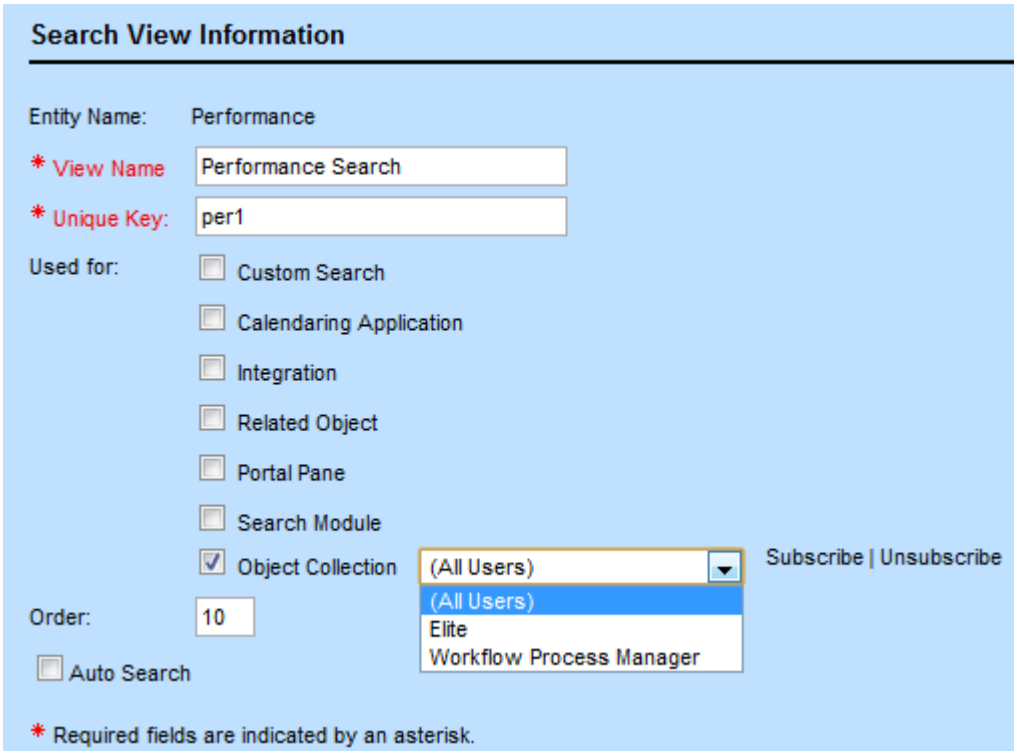
	characters will be replaced by hyphens in the IMAP Client folder names. See Enabling IMAP Search for Custom Object Search Views.
Unique Key	Enter an alphanumeric combination to uniquely identify the search view. Search views within the same object definition cannot have the same unique key, but search views belonging to different object definitions can have the same key. The unique key must be 50 or fewer characters in length and cannot contain spaces, underscores, commas, punctuation marks, or any special characters. After you save the search view, the unique key appears as read-only text and you cannot change it.
Results Display Type 	(Contact and appointment search views only) Select the type of results display for the search screen. The results display type only affects the way the search results appear, not search criteria. Most search views display the search results in the form of a list (also known as a tabular display), with a **Results** tab or a displayed records section that contains the list of results and a **Filter** tab or search criteria section that contains the search criteria. Contact and appointment search views support the following additional display types: • **Card** (for the Contact object)—All results appear in a grid format rather than a list. Each contact's information appears in one cell, instead of in a row across the screen. • **Calendar** (for the Appointment object)—Search results appear in the form of a calendar, with appointments displayed as links to the appointment records. The Appointment object, when invoked from the tab bar in the end user interface, will automatically determine which of its search views uses the Calendar results display type. It will open the Appointments page using that search view. You cannot design display columns for the **Card** or **Calendar** views. However, you can specify how many contact cards appear on a page of results as a system-wide setting.
Used For **Custom Search**	Select this to make the search view a template for a custom search. In a custom search, the user determines which fields (out of the fields available in the search view) are used as criteria. There are several special considerations for a custom search template - for details, see Custom Search Templates.

	Calendaring Application	The use of calendaring applications is not currently supported in TeamConnect.
	Integration	Select to make the search view visible in another application, such as Microsoft Outlook, allowing the user to select an individual record from the list of search results.
	Related Object	Select to make the search view available for searching from within a related object record. In these situations, the object for which you are creating the search view is considered to be a related object. This option is available in search views for all objects except for Contact. If your Filter Display tab uses the search criterion **Parent Project**, you cannot use this **Related Object** feature in the same search view. ***Tip:*** *Create a search view with only **Related Object** selected and set the search view to **Auto Search**. This allows users to quickly see records related to the record they are working with, as the search is performed when they first access the block or tab.*
	Portal Pane	Select to make the search view available to be included in portal panes within home pages. For details, see Creating Search Views for Home Pages. **Card** and **Calendar** results display types cannot be displayed in portal panes. Therefore, when one of these is selected as the **Results Display Type** in a contact or appointment search view, the **Portal Pane** option is removed from the **General** tab.
	Search Module	(Account, Contact, Project, and User search views only) Select to make the search view available in the search module for the object. For more details, see About Search Views in Search Modules. Contact search views display two additional options when you select **Search Module**: • **Person and Company Search—**The search view is available to search for Person and Company contacts in all contact search module system fields except for the Company field in contacts. It is also available in custom fields of type Involved. • **Company Search Only**—The search view is available to search only for Company contacts from the **Company** field

		of contacts and in custom fields of type Involved, but not from any other contact search module system fields.
Used For	**Object Collection**	Select to make the search view available as one of the **Collection** links in the left navigation pane of the application's user interface. When regular users are looking at collections of records, this search view will be one of the links available to them. Clicking on the link will launch this search view. Users do not ordinarily get access to a collection unless they manually subscribe to the collection. However, you can give users access to an object collection as explained in the process Subscribing and Unsubscribing for Collections.
Order		Type an integer to specify the display Order in which you want this search view to appear in the **Current View** drop-down list on the search screen, relative to other search views defined for the selected object. The search view with the lowest order is the first search view in the list, and appears when the user first opens the search screen. This is also the search view used by Global Searches for specific object records. If search views have the same display **Order** (for example, they are all set to 0), they are sorted alphabetically in the **Current View** drop-down list.
Auto Search		Select this check-box to automatically do a search and display search results based on predefined criteria on the **Filter Display** tab. For more details, see About Auto Search.

1.1.7.1.3.1 Subscribing and Unsubscribing for Collections

When a search view's **Used for** field value is **Object Collection**, you may subscribe users to that collection. As shown in the General Tab on Search View Screens image, when subscribing, you can choose to subscribe all users, or to subscribe users from a specific user group from a drop-down list.



General Tab on Search View Screens

Click the **Subscribe** link to apply your selection. You may also click the **Unsubscribe** link to remove the selected groups from that collection's subscription.

You can select multiple groups at the same time in the drop-down list when you are subscribing or unsubscribing. Additionally, if you add new members to a group, they are automatically subscribed to all assigned search views.

When you select **All Users** and then click a link, you get a warning that subscribing or unsubscribing all users might take a long time. Click **OK** in the warning box to continue the subscription process.

1.1.7.1.3.2 About Auto Search

Auto Search automatically displays results when the user selects the search view. If the search view is the first in the **Current View** drop-down list (according to the display order of the object's search views), the search results appear as soon as the user accesses the search screen.

Note: *Auto search has no effect on search views that have Used For values equal to Custom Search. Such search views will always pause for user edits and wait for the user to click the Search button. An end user, when saving a custom search, can choose whether or not to automatically execute the saved search when it is retrieved.*

Auto search is useful for creating search views that always use the same qualifiers. For example, when a user is working with a project record and wants to see all related expense records, it is helpful to provide a search view that automatically displays the Expense records.

Tip: *Using auto search is not recommended if there is typically a large number of results for the type of search, because the user cannot take any further action until the search is complete.*

Auto search can be useful in the following areas:

- For the default search view of an embedded object.
- For the default search view of a child object.
- For a search view that appears in a portal pane on a home page.
- For a search view that appears in a search module window.

1.1.7.1.3.3 About Search Views in Search Modules

You can set a search view for Contact, Account, User, or any Custom Object (except for embedded Custom Objects) to be available in search module. These are the only objects whose records can be selected in search module fields. Making the appropriate search views available in a search module provides the user with the most helpful options for finding the necessary record.

To make a search view available to the user in search module fields, select the **Search Module** check-box on the **General** tab of the search view.

The following are important points about search views in search modules:

- Search views that you use in search modules should be designed with the needs of the user in mind when he or she is selecting a record in those particular fields. For example, in custom fields of type Involved (which display Contact search views), it may be very important that the user is able to search by contact categories.
- Certain fields must be added as search qualifiers on the **Filter Display** tab when a search view is made available in the search module. This is because these fields are used when a user performs a quick search directly from the search module field.

  The following text strings are the search qualifiers required for search views in each type of search module:

- Contact Person and Company Search: **First Name Upper**, **Last/Company Name**
- Contact Company Only Search: **Company Name**
- Project: **Name Upper**, **Number**
- Account: **Name**
- User: **First Name**, **Last Name** (of contact)
- You cannot nest search modules within search modules. Search views in search modules must not have qualifiers which appear as search module fields, because TeamConnect does not display a search module from within another search module.

  For example, you cannot add the **Vendor** field to the **Filter** tab or search criteria section of an account search view in a search module, because it is a contact search module field.

  Tip: *To verify that your search view is not displaying any search module fields, open the search module and select your search view, and look for any fields with the magnifying glass and folder icons next to them. These are search module fields.*

- In search views for Contact that are available in the search module, you can display the **add person** and **add company** buttons, which allow the user to add contacts directly through the search module. For details, see Search Module Actions in Contact Filter Display.
- The user search module displays search views that you define for the User system object, not the Contact object.
- System fields that appear as search module fields, such as the **Contact** field in an Involved record, display available search views to the user in the following manner:
  - If you make multiple search views available for use in the search module for an object, they become available in the **Current View** drop-down list of the search module for the user. The search views are sorted according to their **Order**. The first search view in the list appears in the search module by default.
  - If you only make one search view for an object available in the search module, then it appears automatically in the search module window where users access it through system fields.
  - You cannot specify a particular search view to be displayed for each individual system field.
- In custom fields that appear as search module fields, you have the following options:
  - You can specify that only one specific search view appears.
  - You can allow the user to select from all available search views in the **Current View** drop-down list, if desired.

  For details about using search views in custom fields, see:
  - Search Views for Involved Custom Fields
  - Search Views for Custom Object Fields

1.1.7.1.3.4 Enabling IMAP Search for Custom Object Search Views

Email messages can be shared between IMAP clients and TeamConnect Documents folders for custom object records (such as Dispute attachment folders in the Documents area). You must enable an IMAP Search option for a custom object definition's search views before corresponding TeamConnect-IMAP Mail folders for that search view's results become available in the IMAP client.

Tips

Keep the following tips in mind when enabling IMAP search for custom object search views:

- Only custom object search views may be configured to display IMAP folders.
- If a custom object has no search views with IMAP Search enabled, the corresponding custom records will not display as TeamConnect-IMAP Email folders from the IMAP client or TeamConnect views.
- If a custom object search view with IMAP enabled returns no record results, the record's TeamConnect-IMAP Email folder still displays.
- The current user only sees results from a custom object search view corresponding to records that the user has rights to see.

- Special characters (?, \, /, |, :, <, >, *, %, #, &, (,), {, }, [,], _, ") in custom object search view names or **Record Number** fields are replaced with hyphens (-) in the IMAP client view when IMAP Search is enabled.
- When using Lotus Notes as an IMAP client, do not to use underscores (_) in search view names, project numbers, or IMAP subfolders because Lotus Notes removes the first underscore from the folder names.

To enable IMAP Search for Custom Object search views

1. From the **Go to** drop-down list, select **Object Definitions**.
2. Click the custom object (for example, Dispute).
3. Select the **Search Views** tab.
4. Click the Search View for which to create an IMAP folder.
5. From the **General** tab, under **Used for**, check **IMAP Search**.

Creating an IMAP Search View

6. Click **Save**.

   The result from the IMAP client view is that users who have rights to view records for that custom object are able to view IMAP folders for the record's search views in the following format:

   AccountName

   |----*ObjectType*

   |----*ViewName*

   |----*RecordNumber:PrimaryKey*

   Where:

   - *AccountName* is the TeamConnect IMAP account name.
   - *ObjectType* is a custom object type, for example Disputes.
   - *ViewName* is the name of a Search View, for example MyDisputes.
   - *RecordNumber:PrimaryKey* is the format of the record ID, for example DISP_0007:57 or DISP_0010:58.

1.1.7.1.3.5 Projects and Third-party Calendars

Information about appointments and projects can be shared between TeamConnect and third-party calendaring applications, for example, Microsoft Outlook. When a user creates a TeamConnect appointment from a third-party calendaring application, the appointment first needs to be associated with a TeamConnect project.

From the TeamConnect project object definition, you must enable a search view type, **Calendaring Application**. Afterward, the project becomes available for selection from the third-party calendaring custom form for TeamConnect appointments.

To publish TeamConnect projects to Third-party Calendaring Applications

1. From the **Go to** drop-down list, select **Object Definitions**.
2. Click the custom object (for example, Dispute).
3. Select the **Search Views** tab.
4. Click the Search View to use to filter available projects for association with appointments created in 3rd party calendaring applications.
5. From the **General** tab, under **Used for**, check **Calendaring Application**.
6. Click **Save**.
7. The result from Microsoft Outlook is when users create appointments using the custom TeamConnect form, they are able to select TeamConnect projects to associate with the appointments.

1.1.7.1.3.6 Custom Search Templates

Any search view can be designated as a custom search template by setting its **Used For** value to **Custom Search**. This allows an end user to retrieve the template's information and use it to design an individualized search at runtime.

TeamConnect automatically creates a custom search template at the moment that a newly created object definition is saved. This default template contains most system fields, plus most of the custom fields that belong to the object definition's default category. If you change an object definition later to include more custom fields, you should revise the custom search template to contain the new fields that you want to allow to be searched.

It is strongly recommended that there be only one custom search template in existence at one time for a given object definition. When more than one template exists, and a user requests a custom search, the template that was created earliest will be chosen. If you wish to designate a search view as a template, and there is already an existing template, it is recommended that you first edit the search view that corresponds to the existing template, and uncheck every option in the "Used for" field of the **General** tab. That search view then will not be visible to custom search or any other operations. Then you may designate the new search view as a custom search template.

When a search view is used as a template, it is required that all of the "Used for" options, except **Custom Search**, be unchecked.

Columns that are defined in the **Filter** tab have an extra option, the **Include by Default** check-box. If checked, this means the column will appear as one of the choices that may be used as a criterion in a custom search. The end user may override this setting at runtime.

Columns that are defined in the **Results Display** tab have an extra option, the **Include by Default** check-box. If checked, this means the column will appear in the results of a custom search. The end user may override this setting at runtime.

A search view used as a template does not need, and will not use, Section Title information that is entered in the **Filter** tab of the search view definition.

Note: *For Custom Search templates, it is important that each column in the Filter tab, and each column in the Results Display tab, has a unique value in its "order" attribute. For example, if you give two Filter columns the same value in "order", only one of those columns will appear at runtime to the end user.*

1.1.7.1.4 Search View Results Display

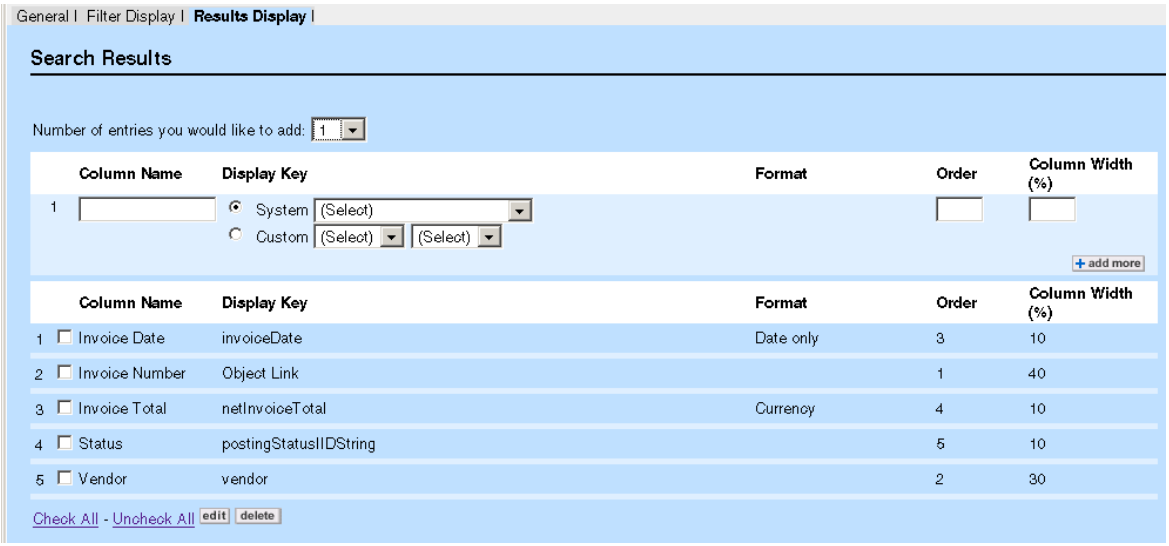
The **Results Display** tab of the Search View screen allows you to specify how an object's search results appear to users. The displayed records section or **Results** tab can display the content of fields from each record that meets the search criteria. For example, search results could display each record's name, number, and the main assignee, as shown in the Displayed Records Example image.

You can set search results to display as vertical columns or as horizontal columns that span the width of the results display. You cannot set all columns to span - at least one column must remain vertical.

Note: *The **Results Display** tab is not visible for search views whose **Used For** selections (in the Filter Display tab) are limited only to "Filter".*

Search results provide links to the listed records. The link that users click to open each record is called the object link. One record is listed on each line of results.

The object link always appears automatically in the search results. On the **Results Display** tab of the search view, you specify the layout of the other columns (not rows) that display the information for each record, as shown in the following image



Results Display Tab with a Display Columns List under Display Key

Note: *If the search view is for the Contact or Appointment object and you have selected* ***Card*** *or* ***Calendar*** *in the* ***Results Display Type*** *drop-down list on the* ***General*** *tab, you cannot modify the way results appear.*

The appearance of search results is also affected by system-wide settings and each individual user's **Preferences** settings.

The following table describes the items on the **Results Display** tab.

Results Display Tab

Field or column	Description
Column Header	Type a label (maximum 250 characters) that describes to the user the record information that appears. This label appears at the top of the column in the displayed records section or the **Results** tab of the object's search screen.
Display Key	Select system or custom fields as display keys for the results of the search screen. Display keys help users identify records returned by the search. The values in these fields are pulled from the database and appear in the search results on the object's search screen, giving users a summary or "preview" of each record. To specify a display key, select one of the following elements:

	• **System**—Allows you to select a system field from the drop-down list.
	  ○ When you select a system field that links to another object table in the object model, an Object Navigator field appears. You can use it to add attributes of related objects and sub-objects as display keys. For example, in an Invoice search view, you can include the **Service Date** field from a line item. For more details about using Object Navigator, see Using Object Navigator.
	  ○ When you select a system field that can return multiple values for a record, another drop-down list appears. You can use this drop-down list to be more specific about the values that display in the search results.
	• **Custom—**Allows you to select a category and a custom field name from two drop-down lists.
	When you select a custom field of type Custom Object or Involved, an Object Navigator field appears. You can use it to add custom fields from other custom objects or Involved records as display keys. For example, if Claim is a child of Policy, you can use Policy custom fields from any category as display keys in a Claim search view.
	• **Related Object—**Allows you to select a child or embedded object from the drop-down list.
	After you select an object, click the Object Navigator icon, and select an attribute of that object to be a display key.
	If you want users to access records directly from the search results, one of the display keys must be defined as an object link. See Defining Object Links.

Format	Specify a format for number, category, or date fields that appear in the search results. When selecting a date field, you can select **Date and time** or **Date only**. If you do not select a date format, the date appears in date only format. When selecting the **Default Category**, **Categories**, **Default Role**, or **Roles** field, **Category Description** is the default format, which means the category or role name appears in the search results (for example, L110). If you specify **Category Full Hierarchy**, the full hierarchy of the category or role appears in the search results (for example, Outside Counsel Fees > Litigation Code Set > L100 > L110). When selecting a field that displays a number, you can specify **Currency**, **Decimal**, or **Percent**. If you select **None** or make no selection, the number appears without any formatting.

	Important: *The* ***Currency*** *format uses the symbol for the system default currency to display the number values in the search results, even if a particular numeric value is actually in a different currency.*
Order	Type an integer for the order in which this column is to be displayed, lowest to highest being displayed from left to right, in the search results. ***Warning:*** *If multiple columns have the same display order value, only one will appear at runtime. Be sure to use unique values for* ***Order****.* ***Tip:*** *If you set the display* ***Order*** *for your columns in increments of 5, you can then easily make slight modifications to the order when necessary without having to recreate the order structure.*
Span?	If checked, the selected column is displayed as a horizontal field that spans the width of the results display. The appearance of spanned columns depends on the information in the column. The span feature applies to the search result pages for the following: • Global Search • Recently Viewed Items • Workflow Approvals • Custom Search • Related Objects • Portal Panes • Object Collections ***Note:*** *You cannot set all columns to span—at least one column must remain vertical. Additionally, if you need to sort information in a results display, spanned columns are not included in the sort.*
Column Width (%)	Enter the percentage for this column out of the total width of the results display. If the total of all column widths of the defined results display columns does not equal 100 percent, then you no longer have control over the column widths. They are resized automatically based on their contents, browser behavior, and other factors.

	To allow the columns to be sized automatically according to the contents, leave this field blank when defining all columns in the results display.
Display Columns List	Displays a list of all columns that have been defined for the search view results. To edit or delete display column definitions, select the check-boxes next to the display column definitions and click **edit** or **delete**. Items in the **Display Key** column cannot be edited. If the qualifier was selected with Object Navigator, the first 90 characters of the Object Navigator path appear in the Attribute column. For details on paths, see Using Object Navigator.

1.1.7.1.4.1 Displayed Records Section Examples

Object Search Screen

The display column definitions shown in the Results Display Tab with a Display Column List under Display Key image produce the displayed records section of the object's search screen, as shown in the following image. The Results Display tab also contributes to the following parts of the image:

- The column headers and display order of the columns.
- The object links that allow the user to link directly to each record that matches the search criteria.
- Display keys that translate into specific information about each record that helps the user to identify it.

Displayed Records Example

Results Display with Spanned Columns

Date	Type	Category	TK	Rate	Units	Disc	Adj	Amt
Mar 1, 2011	Fax	L310 Written Discovery	Lee, Deanne	£500.00	0.20	£0.00	£0.00	£100.00
Description: Attention to e-mails and correspondence regarding discovery, new trial date.								
Mar 11, 2011	Fax	L320 Analysis/Strategy	Lee, Deanne	£335.00	0.60	£0.00	£0.00	£201.00
Description: Email correspondence with persons in charge regarding positive results from grand jury; email correspondence regarding same; telephone conference with								
Mar 11, 2011	Fax	L310 Written Discovery	Lee, Brendon	£335.00	0.20	£0.00	£0.00	£67.00
Description: Reviewed verifications received; drafted and revised letter to Plaintiffs' counsel enclosing verifications.								
Mar 14, 2011	Fax	L160 Settlement/Non-Binding ADR	Lee, Brendon	£500.00	0.40	£0.00	£0.00	£200.00
Description: Attention to e-mails regarding proposed mediators, scheduling issues.								

Spanned Columns Example

1.1.7.1.4.2 Defining Object Links

The *object link* is an essential component of search views that is designed to provide users access to records. It provides a hyperlink on the displayed records section that allows users to link directly to records that are retrieved by the search. Without an object link, records appear on the displayed records section of the object's search screen, but users cannot access them.

However, object links are not necessary in search views that are designed for the purpose of creating reports. For example, you might want to use a search view for creating a list of all currently open Matter records, and you do not need a link to records in the resulting list.

Object links appear as the name, description, or number for the record, depending on the object. See the following table for a complete list of which field appears as the object link for each object.

Object Links in Search Views

Object		Field displayed as object link
Account		Name
Appointment		Subject and Location
Contact	**Person**	Last name, First name
	Company	Company Name
Contact Group (Address Book)		Group name
Document		Name
Expense		Description
Group		Group Account

History	Description
Invoice	Invoice number
Project (custom object)	Name, Number, or both, depending on the setting of the **Auto Numbering Pattern** on the **Unique ID** tab of the object definition.
Task	Subject
User	Last name, First name [of contact record]

1.1.7.1.4.3 Creating Object Links

To create an object link for a search view

1. Open the appropriate object definition, and then click the **Search Views** tab.
2. Select the appropriate search view.
3. Open the **Results Display** tab.
4. Select **System**.
5. In the drop-down list, select **Object Link**.
6. Complete the rest of the fields as indicated in the Results Display Tab table.
7. Click **add more**.

   The object link is added to the **Display Columns List**.
8. Click **Save** to save the object link for the search view.

   The next time a user opens the search view in a search screen, this object link appears on the displayed records section or the **Results** tab of the object's search screen.

1.1.7.1.4.4 Creating Results Display Columns

To create display columns for a search view

1. Open the appropriate object definition, and then click the **Search Views** tab.
2. Select the appropriate search view.
3. Open the **Results Display** tab.

4. Select the **Number of entries you would like to add** from the drop-down list.
5. For each data entry row, complete all appropriate fields as described in the Results Display table.
6. Click **add more** to add the columns to the list and continue adding more display columns. Otherwise, click **Save**.

   The next time a user opens this search view, the display columns appear on the displayed records section or the **Results** tab of the search screen.

1.1.7.1.5 Search View Filter Display

Search qualifiers appear as criteria on the **Custom Search** page to allow the user to enter search criteria when conducting a search. You need to create a qualifier for each field you want to include in the search view's search criteria. The more qualifiers you add when creating a search view, the more types of information users are able to search by to find the records they need.

You can define search qualifiers and the appearance of the **Custom Search** page on the **Filter Display** tab of the search view.

You can add the following items to the search view on the **Filter Display** tab:

- Visible qualifiers (see Visible and Invisible Qualifiers)
- Invisible qualifiers (see Visible and Invisible Qualifiers)
- Section Titles (see Creating Section Titles)

Note: *If no qualifiers are specified in the search view, then no records are filtered out of the search results. Instead, all records for the object are listed.*

1.1.7.1.5.1 Visible and Invisible Qualifiers

Search qualifiers are usually displayed as search criteria fields on the **Filter Display** tab of the search view. These qualifiers are called visible qualifiers, and are available to the user. For example, in a search view for Invoice records, a user could search by **Invoice Date**, **Posting Status**, **Invoice Number**, and the **Service Date** of a line item. To make a qualifier visible, select the **Visible** check-box when defining the qualifier on the **Filter Display** tab of the search view.

However, in certain search screens, you might want to limit the scope of the user's search results using qualifiers that are not available on the **Custom Search** page. This is done by creating invisible qualifiers. Invisible qualifiers cannot be accessed by the user, because they are not displayed in the search criteria. An invisible qualifier is applied to all searches performed with this search view.

For example, you might want users who search for Matter records to see only records they are assigned to in the search results. This would require creating an invisible qualifier that has the field name assigneeList_User and the condition value Current User (see the search qualifier listed in the Filter Display Tab on Search View Screens image). Only records that are assigned to the user who is currently performing the search appear in the search results.

The qualifier with the label **Created this month** in the Filter Display Tab on Search View Screens image is another example of an invisible qualifier. This qualifier automatically limits the search results

so that only records that have been created during the last 30 days appear on the **Results** page of the user's search.

To make a qualifier invisible, clear the **Visible** check-box when defining the qualifier on the **Filter Display** tab of the search view.

Important: *If no condition value is entered, an invisible qualifier has no effect.*

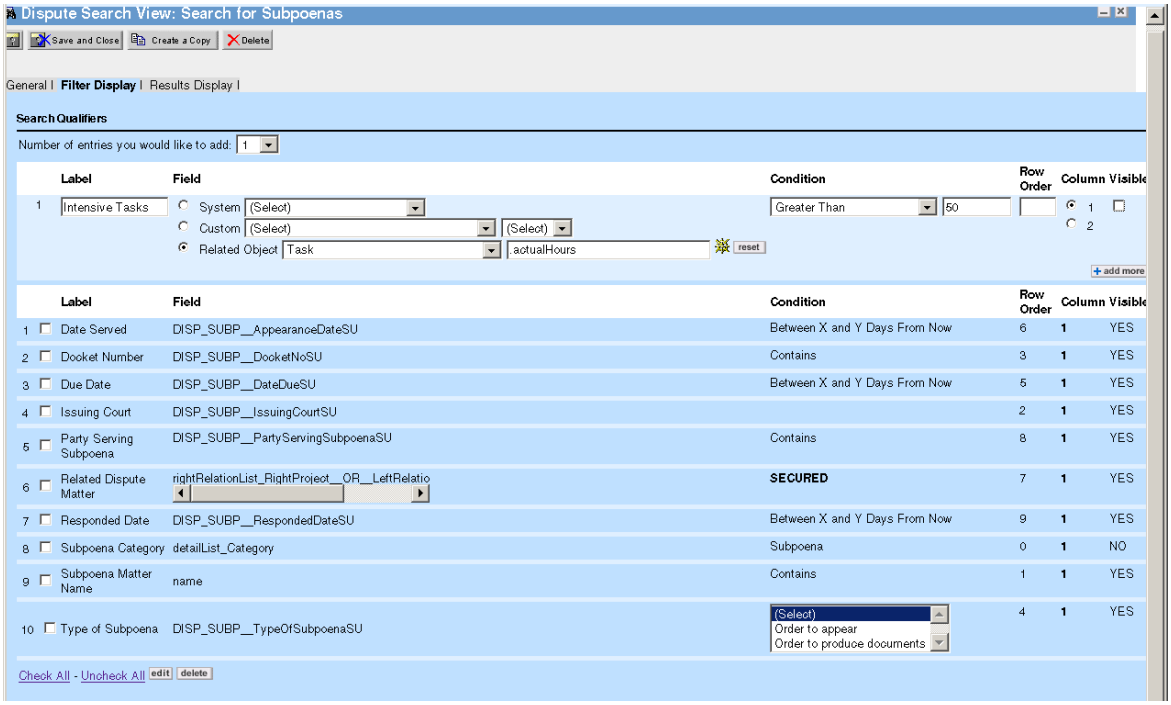
1.1.7.1.5.2 Qualifier Conditions

Qualifier conditions are composed of two parts: an operator and a value.

- The operator lets the user specify what kind of limit to use when searching. For example, operators for a date field include **More Than X Days Ago** and **Between Dates**.
- The value is used with the operator. For example, you can enter **5** for the number of days in a date field search.

When creating a search view, the **Condition** fields on the **Filter Display** tab give you the ability to select which operator should appear by default when the user accesses the search criteria of the object's search screen. You might use this function, for example, when you want the **Contains** operator to be displayed by default for a Text custom field.

If needed, you can also select a default value for the operator, such as the name of a user, a number of days, or a dollar amount. Typically, you need to select a condition value only when creating an invisible qualifier. However, you can provide default selections in a visible qualifier as well. If you select a default operator or value for a visible qualifier, then the user is able to make different selections if desired. The following image shows examples of conditions.



Filter Display Tab on Search View Screens

Filter Display Tab on Search View Screens

Field or column	Description
Label	Type a name (maximum 250 characters). This name appears as a label for the field on the **Filter** tab or search criteria section of the search screen. It indicates to the user what information to enter in the field for the search criteria. ***Note:*** *A colon (:) is automatically placed after the label text in the search view.*
Field	Do one of the following to select a field that will appear on the **Filter** tab or search criteria section of the search screen as a search qualifier: • **System**—Allows you to select a system field from the drop-down list. When you select a system field that links to another object table in the TeamConnect object model, an Object Navigator icon and field appear. You can use these to add attributes of related and sub-objects as qualifiers. For example, in an account search view, you can include the company of the involved party who can post transactions to the account. For more details, see Using Object Navigator. You can also select **System** to create a section title that appears on the **Filter** tab or the search criteria section of the search screen. For details, see Creating Section Titles. Upon clicking **System**, you are given the option of how to view the field on the **Filter** tab or search criteria section of the search screen: ○ Drop-down list ○ Multi-select list Choose the option that fits best with your user interface. **Custom**—Allows you to select a category and a custom field name from two drop-down lists. When you select a custom field of type Custom Object or Involved, an Object Navigator field appears. You can use it to add custom fields from other custom objects or Involved records as qualifiers. For instance, if Claim is a child of Policy, you can use Policy custom fields from any category as qualifiers in a Claim search view. For many fields, you are given the option of how to view the field on the **Filter** tab or search criteria section of the search screen: ○ Drop-down list ○ Multi-select list

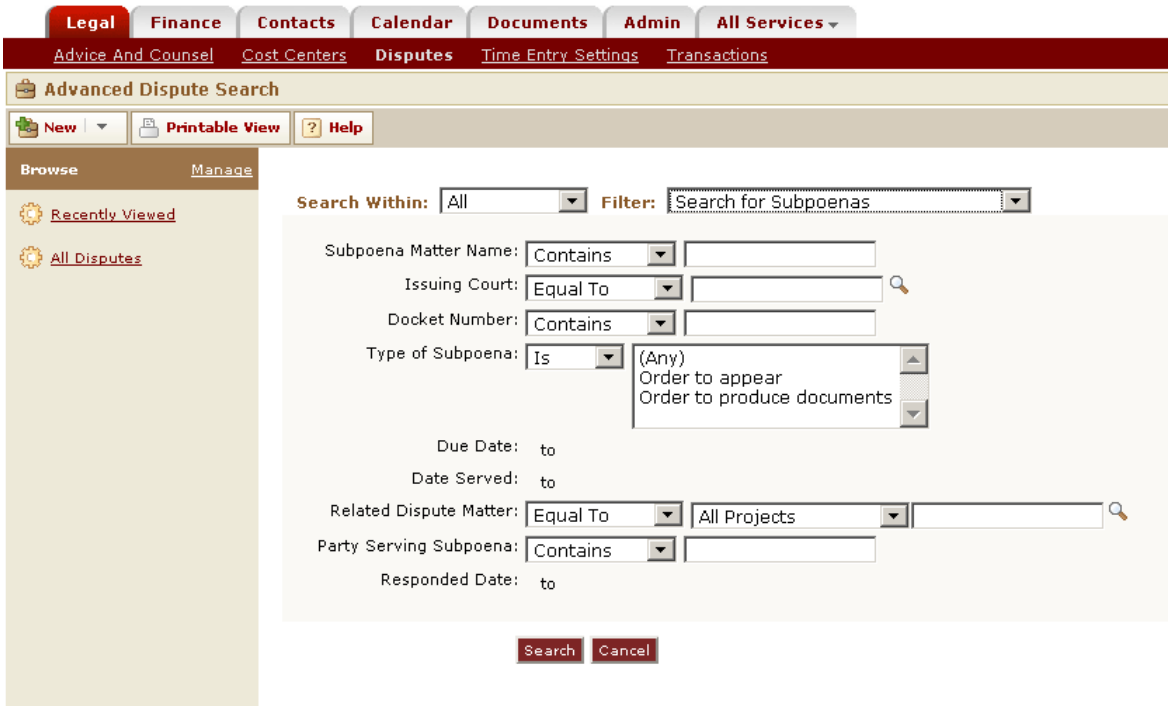
	○ Search field ○ Multi-select search field Choose the option that fits best with your user interface. The "search field" options use auto-suggest; after the user types two or more characters in the field, a list of matching records (up to a maximum of 10) appears next to the field and the user can click on a list entry to select it. The user can also click the search icon to launch a separate page on which they will select a value for the field. **Related Object**—Allows you to select a related, child, embedded, or sub-object from the drop-down list. Then, use Object Navigator to select a field from that object as a qualifier. ***Caution:*** *If the search view is set to be available in the search module, do NOT add search qualifiers that also use the search module, such as certain custom fields of type Custom Object or Involved. These fields will not be displayed properly in the search module window. For more details, see About Search Views in Search Modules.*
Condition	If desired, create a default condition for the qualifier. For most fields, you can select an operator only, or an operator with a value. If you do not specify an operator or value in the condition, and the **Visible** check-box is selected for the qualifier, then the default operator appears in the search criteria of the object's search screen. The user is able to set the operator and value as desired.
Row Order	Type an integer for the **Row Order** in which the qualifiers appear. Qualifiers are listed vertically in the search criteria of the object's search screen, using the order you specify. The display order should be useful for users when they are entering search criteria. ***Warning:*** *If multiple qualifiers have the same row order, only one will appear at runtime. Be sure to use unique values for Row Order.*
Column	Select the column in which the search qualifier will be listed in the search criteria of the object's search screen. Choose Column 1 to display the qualifier in the left column of the screen; choose Column 2 to display the qualifier in the right column of the screen. The ability to choose which column the search qualifier filter will reside in gives you greater control of the screen elements and offers the ability to manipulate the layout of the object's search screen.
Visible	Select whether the qualifier for the search view is visible. If the qualifier is not visible, users do not see the field and are unable to enter criteria

	in this field on the **Filter** tab or the search criteria section of the object's search screen. ***Important:*** *Invisible qualifiers must have a condition value.* For details, see Visible and Invisible Qualifiers.
Search Qualifiers List	Displays a list of existing qualifiers for the search view. Select the check-box next to a search qualifier and click **edit** or **delete** to modify any of its items—except for the **Field** column, which cannot be edited. If the qualifier was selected with Object Navigator, the first 90 characters of the Object Navigator path appear in the Attribute column. For more details on paths, see Using Object Navigator.

1.1.7.1.5.3 Filter Tab Example

The search qualifiers shown in the Filter Display Tab on Search View Screens produce the **Filter** tab or the search criteria section of the object's search screen, as shown in the following image. Notice the section titles and the vertical display order of the search qualifiers. Also notice that the invisible qualifiers are not displayed.

When list fields that are associated with a lookup table are added as search qualifiers, the fields appear as combo list boxes to allow for multiple selections.



Filter Tab Example with Visible Search Qualifiers

1.1.7.1.5.4 Search Module Actions in Contact Filter Display

The contact search module provides users the ability to add a new contact directly through the search module window. If you want users to create contacts this way, you can indicate this by using the settings in the **Search Module Actions** section on the **Filter Display** tab of the contact search view, as shown in the following image.

Search Module Actions Section of Contact Search View

The **Actions** section is only displayed if you have indicated on the **General** tab that the search view should be available in the search module. It appears below the **Search Qualifiers** section (as shown in the Filter Display Tab on Search View Screens image) and provides the options for displaying the **add person** and the **add company** buttons.

If **Company Search Only** is selected on the **General** tab of the search view, then you do not have the option of displaying the **add person** button to the user. The search view is available to users only through the **Company** field of a contact record, which is intended for finding and selecting only company contacts.

If you provide the capability to add a contact through the search view in the search module, consider the following information:

- The **add person** and **add company** buttons appear for the user only when the search view appears in the search module. If the search view is set to be available in other areas, such as search modules or portal panes, the buttons do not appear when the search view appears in those areas.
- When a user adds a contact through the search module, only certain values that the user can provide on the **Filter** tab or the search criteria section of the object's search screen are saved in the new contact record, even if additional fields are available on the **Filter** tab or search criteria section. For details on adding contacts from search modules, see the User Guide.
- If there are any required custom fields for contacts, users will not be able to add contacts through the search module when the category for those custom fields is indicated as the default category of the contact they want to add.
- Any rules that are triggered on the creation of a contact record will also be triggered when the user clicks **add person** or **add company**.

The following table describes the items in the **Search Module Actions** section on the **Filter Display** tab of the contact search view.

Search Module Actions Section of Contact Search View

Field		Description
Allow user actions	**Add Person**	Select to add the **Add Person** button to the Contact search screen. This button allows users to add a person's contact record using the values entered on the

		Filter tab or the search criteria section of the search screen. If **Company Search Only** is selected on the **General** tab of the search view, then this option is not available on the **Filter** tab or the search criteria section of the search module.
	Add Company	Select to add the **New Company** button to the Contact search screen. This button allows users to add a Company contact record using the values entered on the **Filter** tab or search criteria section of the search module.

1.1.7.1.5.5 Creating Search View Qualifiers

Before creating search view qualifiers for your search screen, make sure to plan which qualifiers you want to include in the search view, including visible qualifiers, invisible qualifiers, and section titles. This helps you set the correct display order for each qualifier.

Search views return all subcategories in the hierarchy of a category search. Because child categories contain their parent's category, a search that returns a parent category shows all of its child items as well.

To create search view qualifiers

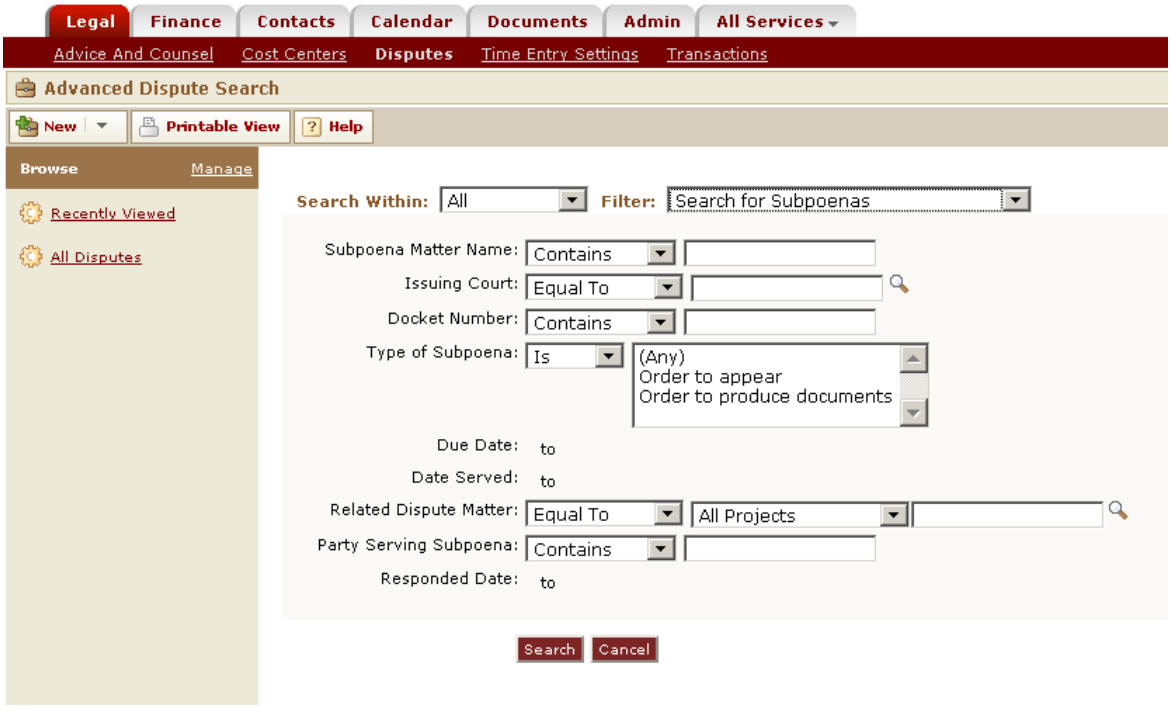
1. Open the appropriate object definition, and then click the **Search Views** tab.
2. Do one of the following actions:
   - To change an existing search view, click the name of the search view to open it.
   - To create a search view, click **new** and do the following actions:
     i. Complete the **General** tab. For details, see Defining General Search View Information.
     ii. On the **Results Display** tab, add at least one column to the results display. This allows you to save the search view. For details, see Search View Results Display.
     iii. Click **Save** to save the search view.

     After the search view is saved, you can define search qualifiers.
3. Click the **Filter Display** tab.
4. In the data entry rows, enter the appropriate information as described in the Filter Display Tab on Search Views Screens table.
5. Click **add more** to continue adding qualifiers. Otherwise, click **Save** to save the qualifier as part of the search view.

You have added search qualifiers to the filter display of the search view. The next time a user opens the search view in a search screen, these search qualifiers appear on the **Filter** tab and/or the search criteria section of the object's search screen.

1.1.7.1.5.6 Creating Section Titles

You can use section titles to label groups of the search qualifiers in your search views. For example, if you have a lot of qualifiers for date fields in the search view, you can organize your qualifiers into two sections entitled "General Record Information" and "Matter Record Dates," as shown in the following image.



Filter Tab Example with Visible Search Qualifiers

Note: *Section titles are ignored for search views that are used as custom search templates.*

By default, section titles appear in size 10 black font, but users can set their formatting preferences in their **Appearance** settings.

To create a section title in the filter display

1. Open the appropriate object definition, and then click the Search Views tab.
2. Do one of the following actions:
   - To change an existing search view, click its hyperlink to open it.
   - To create a search view, click **new** and do the following actions:

i. Complete the **General** tab. For details, see Defining General Search View Information.

ii. On the **Results Display** tab, add at least one column to the results display. This allows you to save the search view. For details, see Search View Results Display.

iii. Click **Save** to save the search view.

After the search view is saved, you can create a section title.

3. Click the **Filter Display** tab.
4. In the data entry row, in the **Label** field, enter the text of the section title.
5. From the **System** drop-down list, select **(Section Title)**.
6. In the **Row Order** field, enter the display order of the section title.

Tip: *Place the section title in the correct order in relation to the fields for which it is a heading. For example, if the* ***Row Order*** *is set to 5, then the qualifiers you want displayed below it must have display orders 6, 7, 8, and so on. For an example, see the Filter Display Tab on Search View Screens tab image.*

7. Under Column, check the column (or screen location) where the search qualifier will reside in the **Filter** tab or the search criteria section of the object's search screen:
   - Choose Column 1 to display the qualifier in the left column of the screen.
   - Choose Column 2 to display the qualifier in the right column of the screen.
8. Check the **Visible** check-box so that the section title appears in the search screen.
9. Click **add more** to continue adding search qualifiers or click **Save** to save the section title as part of the search view.

The next time a user opens the search view in a search screen, this section title appears on the **Filter** tab or the search criteria section.

1.1.7.1.5.7 Tips for Working with Qualifiers

This section provides more information about how selected search view qualifiers behave in TeamConnect.

Parent Project

When working on a search view for a child object, the qualifier **Parent Project** is available, except when the **Used for** general option in the search view includes the **Related Object** choice.

Assigned On, Unassigned On, and Main Assignee Assigned On

These three qualifiers appear in the **System** drop-down list in the **Filter Display** tab of a search view definition. You can use these qualifiers when building search views against objects that have lists of assignees.

These qualifiers accept values that are date ranges. The behavior of date ranges for **Assigned On** and **Unassigned On** varies, depending on whether the search request also specifies an assignee user ID:

- If you select a specific assignee, then the date ranges will only look at assigned and unassigned dates for that specific assignee.
- If you do not select a specific assignee, then any assignee whose assigned and/or unassigned dates match your date ranges will cause the record to qualify the search results. (If nine out of ten assignees do not qualify based on the date ranges, but the tenth one does qualify, the record will qualify for the search results.)
- If a single assignee appears twice in a record's assignee list, in two different roles, both roles' dates will be examined to see whether either of them qualify to be included in the search results.

The behavior of the date range for **Main Assignee Assigned On** varies, depending on whether the search request also specifies a main assignee user ID:

- If you select a specific main assignee, only records matching that main assignee will be considered, and the date range will only look at assigned dates for those records.
- If you do not select a specific main assignee, then any main assignee whose assigned date matches your date range will cause the record to qualify for the search results.

It is important to remember that TeamConnect only checks dates for the current main assignee. If there was a different main assignee in the past, that information is not known to the search filter. Also, the assignment date for the main assignee is the date that the person was first added to a record. If you added someone as a regular assignee at first, then promoted them to main assignee one month later, the **Main Assignee Assigned On Date** will be considered the earlier date, not the one a month later.

As an example, if you specify a main assignee of "Margaret Thatcher", then use an **Assigned On** date range of August 1st through August 31st, and leave the regular assignee drop-down list set to **ANY**, you will get all records for which Margaret Thatcher is the main assignee, and for which anyone was assigned during August. This does not necessarily mean that Margaret Thatcher was assigned during August-she could be assigned to a record in June and, if someone else was assigned during August, the record would qualify for inclusion. If you want to make your filter more specific, and find records where Margaret Thatcher is main assignee and she was assigned in August, you must use the **Main Assignee Assigned On** date range, not the **Assigned On** date range.

Searching the Documents of an Individual Project (Custom Object) Record

When using the **Search** button from within a specific record's **Document** screen, the qualifier **Search in Folder** presents several choices in a drop-down list. The only choice that is valid for a search inside a specific record is **Current Folder**.

If you are already within a subfolder of a record's documents when you launch such a search, the search results will be limited to that subfolder.

Searching History Records of a Specific Project or a Specific Custom Object Type

Qualifier **Parent Project**, a choice in the **System** drop-down list in the **Filter Display** tab, can restrict history searches to a specific custom object type and/or a specific project name or number.

- When you add the **Parent Project** qualifier to a new or existing search view, the **Condition** drop-down list will show a list of all custom object definitions to which you have "read" authority.
- If you want to restrict this history search to a specific custom object type, choose that type from the drop-down list. Otherwise, choose **ANY**.

When end users choose a search view that contains this qualifier, the field **Parent Project** accepts the number or name of the desired project, and:

- Search results will be filtered to include only projects that match the name or number that the end user enters in that field.
- If the user leaves the field empty, search results will not be filtered by project name or number.
- If you, the developer, chose a specific custom object type when you specified the **Parent Project** qualifier in the search view, search results will be filtered to include only that custom object type, no matter what value the end user puts in **Parent Project**.

Search results will only show records to which the user has at least "read" authority.

For example, assume that you create a search view that includes the **Parent Project** qualifier, and you specify custom object type **Matter** while defining that qualifier. When an end user runs that search view and enters "Turnstone Incorporation" as a value for that qualifier, then only Matter records that match that name will be searched, even if there are other types of custom object records that have the name "Turnstone Incorporation". If, on the other hand, the end user runs the search view but does not enter any value for **Parent Project**, the search will still be filtered to include only Matter records, regardless of what other qualifiers might be used in the search.

Searching by Workflow Status and Workflow Action

Possible values for the **Workflow Status** qualifier include the value **Any** (Workflow status will not be considered when filtering search results) or one of the several possible end states of a workflow, such as Rejected, Error, Pending, Approved, or Canceled.

Field **Workflow Status** is available for the results display for a search view to show the status of specific records. **Workflow Status** lists specific values, such as Pending Approval, Approved, Rejected, Error, and Canceled. This field may be blank if a record has not been submitted to workflow.

Workflow Action

The **Workflow Action** qualifier presents a list of values that vary by object type. Any workflow action that can be defined in an Approval Rule for an object will also appear in the **Workflow Action** qualifier drop-down for that object. For example, the Invoice object allows actions of Post, Void, and Delete. In addition to the actions that are specific to an object type, the qualifier drop-down includes the value **Any**, which means that the qualifier is ignored in the current search.

For example, using this qualifier means that you could limit an Invoice search to searching only for records that are pending approval for posting, not pending approval for void or deletion.

Field **Workflow Action** is available for the results display for a search view to show more detailed status of specific records.

For example, without the **Workflow Action** field the results of an invoice search might show the workflow status of **Pending**. With the inclusion of the **Workflow Action** field the results could say **Pending** and **Void**.

This field may be blank if a record has not been submitted to workflow.

Note: *In cases where there may be multiple workflows open against a single record, only the workflow that was initiated most recently will be evaluated against these qualifiers.*

1.1.7.2 Troubleshooting Search Views

This table explains solutions for possible problems with search views.

Search View Troubleshooting Tips

Possible problems	Possible solutions
The search view states that the search results display is not configurable, and I cannot define the necessary columns to display the matching search results for the end users.	You are likely to encounter this issue in either contact or appointment search views. If on the **General** tab of the search view in the contact or appointment search view, **Card** or **Calendar** display type is selected, you are unable to change the display of search results. To customize the display of appointment of contact search results, select **Tabular** in the **Results Display Type** drop-down list on the **General** tab of the search view.
The **Results** tab or the displayed records section of the object's search screen in the end-user interface fails to have hyperlinks to the records.	Make sure that on the **Results Display** tab of the corresponding search view, the **Object Link** system display key is added.
The column width specified on the **Results Display** tab of the search view fails to take effect on the corresponding **Results** tab or the displayed records section of the object's search screen in the end-user interface.	If the total percentage of all columns does not equal 100%, the system may automatically resize the column to fit the contents, ignoring your settings.

1.1.8 Creating Home Pages and Portal Panes

A *home page* is the first page that users see after logging in to TeamConnect. It is the starting point for all users and can be customized to fit either individual or group user needs. The core functionality of a home page within TeamConnect is to display portal panes, which contain the actual content for the page.

You can create as many different home pages as necessary for your users. Each user can have access to multiple home pages based on which ones you give them access to as members of a group. Each home page that a user has access to is listed on their **Home Pages** menu.

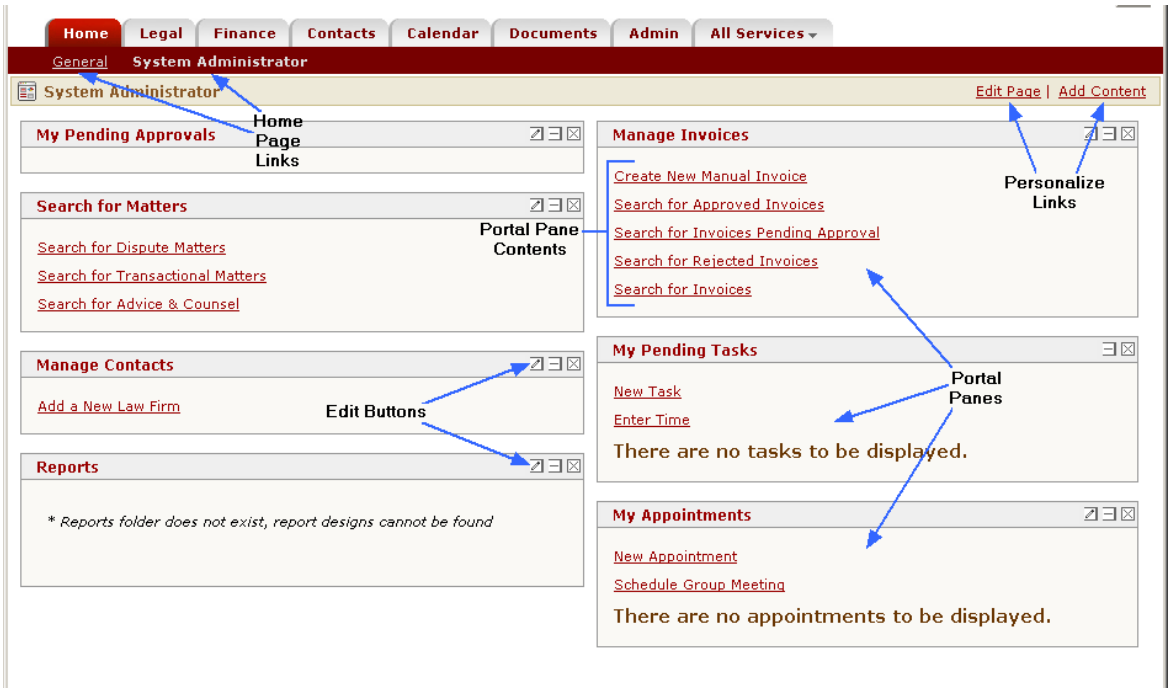
Portal panes are the building blocks of a home page and contain specific content (information) that you define. This content is, in one way or another, beneficial to a user's job performance. Examples

of portal pane content are links to upcoming appointments or helpful web sites, shortcuts to create new contacts, claims, tasks.

1.1.8.1 Home Pages and Portal Panes Basics

To specify home pages and portal panes, you may do the following:

- Create and define multiple portal panes.
- Include a particular portal pane on multiple home pages, as necessary.
- Create portal panes that include one or more types of content (see Portal Panes with Multiple Contents).
- Select the necessary types of content in a portal pane based on its purpose and function as well as the needs of your users.
- Create a common, organization-specific home page that displays corporate logos and messages for users.
- Restrict the availability of a home page to a particular group of users.
- Create a user-customizable home page that displays items such as (but, not limited to):
  - Upcoming appointments and tasks
  - Links to job-related Web sites
  - Stock quotes



Administrator End-User Home Page Components

The following table describes the components of a home page.

User Home Page Components

Field	Description
Home Pages Links	Displays the titles of the home pages available to the user. The user selects the desired home page to view by clicking the desired title. The home page currently being viewed appears in a title bar just below the **Home Pages** links.
Personalize Links	The user can personalize and modify a home page by clicking **Add Content** and **Edit Settings**. Once users modify a home page, they must click **Refresh** to see their changes. You can specify whether to show this menu. For details, see **Allow end users to edit layout only** in the Home Page Settings Tab table.
Portal Panes	Users work with the information the different portal panes. You can add as many portal panes as necessary to a home page. For details, see About Modifying Home Pages.
Portal Pane Content	Users can perform various tasks by clicking links or viewing content results and information within the different portal panes. You can add and create new content for a portal pane. For details, see Portal Pane Contents.
Portal Pane Title Bar	Displays the name of the portal pane. You can display buttons so that users can edit, move, resize, or delete the portal pane. For details about options that you can set for users, see Portal Pane Settings.

Design Checklist

Use the following checklist when creating home pages and portal panes:

- Define the various types of users to be seeing home pages.
- Determine how many home pages are necessary.
- Determine how many portal panes are necessary, with the capability of reusing the same portal pane in multiple home pages.
- Define the content of each portal pane.
- Define home pages and add the appropriate portal panes.
- To make a home page visible to users, select the **Visible** check-box on the **Settings** tab of the home page (shown in the Home Page Settings Tab image).
- Determine whether users are allowed to edit each home page and portal pane or control other options. See Modifying Home Pages and Portal Panes for more details.

Important: *Do not turn on editing options for users until after you are finished designing your new home pages and portal panes.*

1.1.8.2 Portal Panes

As a TeamConnect solution developer, you may define the content and settings of portal panes for your users. By defining the portal panes, you create links to information and control what users can do from their home pages.

Depending on the needs of your users and their job responsibilities, you may select which portal panes are viewable from different home pages. You may either create new portal panes or select the portal panes included with TeamConnect by default.

The explanation of Portal Panes includes the following topics:

- Default Portal Panes
- Opening Portal Panes
- Portal Pane Settings
- Portal Pane Contents
- Creating Portal Panes

1.1.8.2.1 Default Portal Panes

The following portal panes are included with TeamConnect and available for use in home pages:

- **My Appointments—**Displays appointments scheduled for the user.
- **My Documents—**Displays links to files contained in the logged-in user's document folder.
- **My Reports**—Displays a specified report on the user's homepage. From the **My Reports** section, click **Edit Parameters** and select a **Report** from the drop-down list. You can also specify the **Chart Width** and **Chart Height**.
- **My Pending Approvals—**Displays the user's pending approvals.
- **My Approvals—**Displays the user's pending approvals.
- **My Requests—**Displays the user's requests in progress.
- **My Workflows—**Displays the processes that the user is responsible for monitoring. This portal pane only appears to Process Managers.
- **RSS Feeds—**Displays the selected RSS Feed content (for example news feeds).
- **My Stocks—**Displays stock information and links to various companies on the Yahoo Finance page.
- **My Tasks—**Displays tasks assigned to the user.

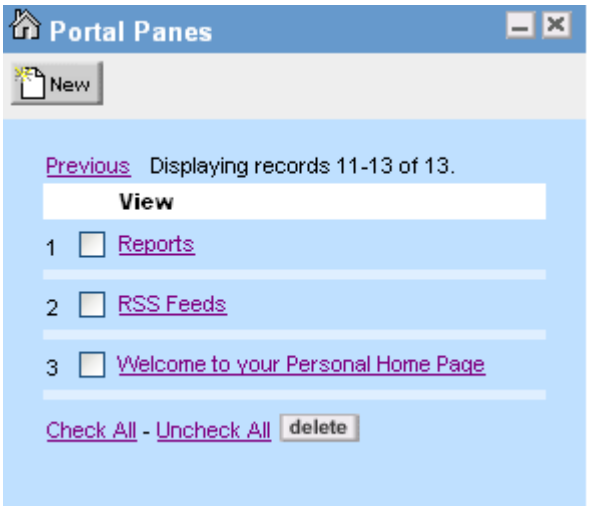
1.1.8.2.2 Opening Portal Panes

To open a portal pane

1. In the **Go to** drop-down list, select **Portal Panes**.

   The **Portal Panes** screen displays a list of portal panes, which include the default portal panes and all previously created portal panes.

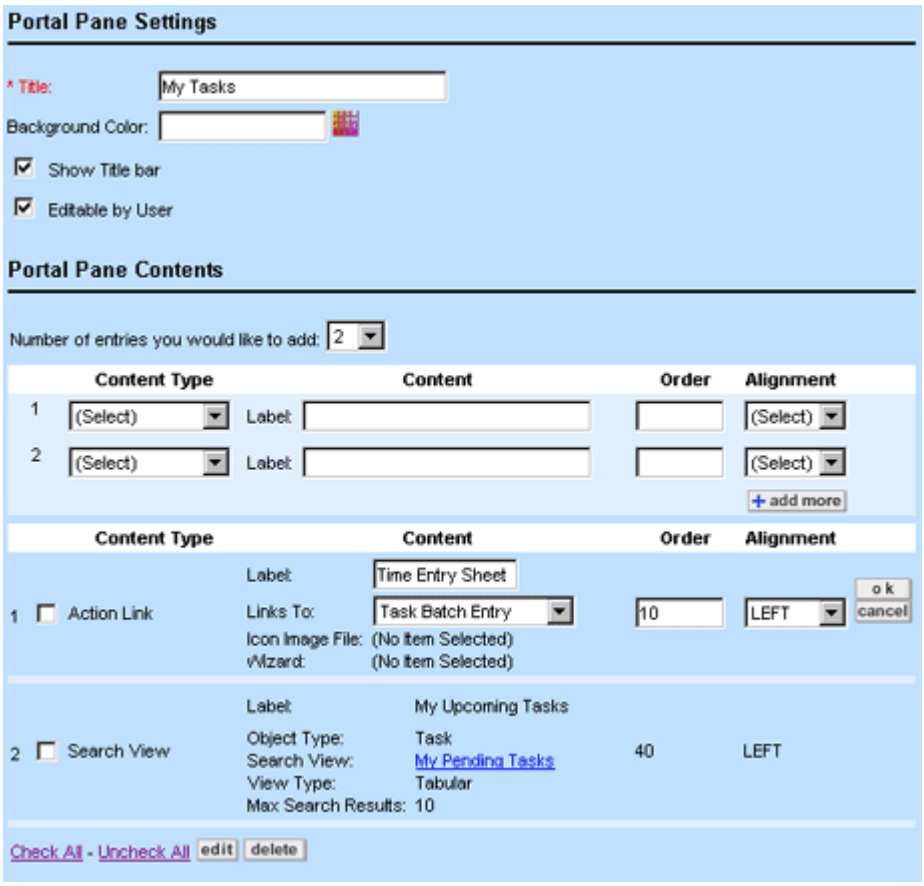
Portal Panes Screen



Portal Panes Screen (continued)

2. From the portal pane list, click the hyperlink of the desired portal pane.

The **General** tab of the portal pane you selected appears.



General Tab on Portal Pane Screens

The **General** tab of the **Portal Pane** screen has the following sections:

- **Portal Pane Settings—**From this section, you may modify general information such as the title and background color of a portal pane. See Portal Pane Settings.
- **Portal Pane Contents—**From this section, you can modify the content that defines how a portal pane functions from a home page. See Portal Pane Contents.

1.1.8.2.3 Portal Pane Settings

The **Portal Pane Settings** section of the **General** tab allows you to:

- Identify the portal pane by giving it a title and a unique identifying key.
- Set the background color of the portal pane.
- Give users specified controls to the selected portal pane.

The following table describes the items in the **Portal Pane Settings** section.

Portal Pane Settings

Field	Description	Comments
Title	Enter a name (maximum 250 characters) that uniquely identifies the portal pane. This name appears in the following locations: • The solution developer's **Portal Panes** screen. • User's **Select Content to Add** pop-up window displayed by clicking the **Add Content** link on their home page. • The title bar of the portal pane.	As long the **Editable by user** check-box is selected, the title bar of the portal pane appears for the user--even if the **Show title bar** check-box is not selected.
Unique Key	Enter an alphanumeric combination to uniquely identify the portal pane content. After you save the settings, you can edit this field only if it is blank. The unique key cannot contain spaces, underscores, punctuation, or any special characters and cannot exceed 50 characters in length.	
Background Color	(Optional) Click the **Color Picker** icon next to the filed and select the desired color from the pop-up window that appears. If no background color is specified for the portal pane, the background color will be the normal background color defined in System Settings.	Search View content items always use the background color defined in System Settings. Custom Content items always use the background colors defined in their Java class files.
Show title bar	Select this check-box to display the title bar of the portal pane. For example: If the appropriate options are turned on, it also displays buttons so that users can edit, move, resize, or delete the portal pane. You can choose which options to turn on.	Changes made by one user from their own home page, do NOT affect the same home page of another user.
Editable by user	Select this check-box to allow users to edit the contents of the portal pane within their home page.	

	The **edit** button is available from the portal pane title bar.	

Portal Pane Setting Dependencies

How the settings of your portal pane take effect depends upon the settings of the home page on which that portal pane appears. The following table shows all the possible combinations of portal pane and home page settings, along with the resulting portal pane functionality:

Home Page and Portal Page Setting Combinations

Home page settings	Portal pane settings	Results				
		For the portal pane, user can				
		See title bar	Edit	Move	Resize	Remove
☑ Visible ◉ Do not allow end users to edit layout or content ○ Allow end users to edit layout only ○ Allow end users to edit layout & content	☐ Show Title bar ☐ Editable by User					
	☑ Show Title bar ☐ Editable by User	X				
	☐ Show Title bar ☑ Editable by User	X	X			
	☑ Show Title bar ☑ Editable by User	X	X			
☑ Visible ○ Do not allow end users to edit layout or content ◉ Allow end users to edit layout only ○ Allow end users to edit layout & content	☐ Show Title bar ☐ Editable by User					
	☑ Show Title bar ☐ Editable by User	X		X	X	
	☐ Show Title bar ☑ Editable by User	X	X	X	X	
	☑ Show Title bar ☑ Editable by User	X	X	X	X	

Visible Do not allow end users to edit layout or content Allow end users to edit layout only Allow end users to edit layout & content	Show Title bar Editable by User					
	Show Title bar Editable by User	X		X	X	X
	Show Title bar Editable by User	X	X	X	X	X

1.1.8.2.4 Portal Pane Contents

To specify what a portal pane displays, you select a content type. The content type determines the information that the user will access (such as tasks) or the action that they will perform (such as scheduling new appointments).

Portal pane content provides users with quick access to important company announcements, various tasks, search results, reports, Web sites, and so on.

The kinds of information that are added to portal panes are referred to as content types. The following table lists the various content types that you can include in a portal pane, along with a description and usage of each.

Portal Pane Content Types and Their Usage

Content type	Description	Use to
Action Link	Allows users to create new records from their home page or to access a TeamConnect user tool from their home page. When a user clicks an action link from a portal pane, the result is the same as selecting the corresponding options from either the **Create a new** or **Tools** drop-down lists in the user interface. The portal pane titled "Home Owner Claims" in the Administrator End-User Home Page Components image is an example of an action link. For more details about action links, see Adding Action Links.	Create the following records with or without a wizard: • Accounts • Appointments • Contacts (can specify a New Company link or a New Person link) • Contact Groups (Address Books) • Custom Objects • Expenses • Invoices • Tasks Access the following tools: • Document Generator

		• Expense Batch Entry • Task Batch Entry • Scheduler
Object Link	Allows users to open specific records or to search for specific records from their home page. When a user clicks an object link from a portal pane to search for a specific record, the result is the same as selecting the corresponding options from the **Go to** drop-down list in the user interface. The portal pane titled "Important Contacts" in the Administrator End-User Home Page Components image is an example of an Object Link portal pane. Also, allow users to open specific search screens from their home page. For details, see Adding Search View Object Links. ***Important:*** *Search View object links function differently from the Search View content type. Search View object links are essentially links to a Filter tab of a search screen, whereas content items of the type Search View display the results of a search within the portal pane.* For more details, see Adding Object Links.	Create links to access the following: • Specific contacts • Specific projects (custom object) • Specific accounts • Specific users • Specific groups • Search views • Specific global forums
Search View	Allows users to view specific search results that appear within their portal pane. Search views in portal panes are the same search views defined for each object within TeamConnect. You can include any existing search view in a portal pane--provided that the **Usable in Portal Panes** option is selected when you create the search view.	Display search results for a specific search view within the portal pane in one of the following ways: • **Tabular view**—Search results appear in a table format, like the **Results** tab of most search screens.

	Tip: *Create separate search views for portal panes.* To learn more about Search View qualifiers, see Search View Filter Display. The portal pane titled "My Upcoming Appointments" in the Administrator End-User Home Page Components image is an example of a search view. For more details, see Adding Search Views.	• **Drop-down list**—Search results appear in a drop-down list.
Report	Allows users to see a report's results, or to link to a report. The reports must already have been designed and tested. A report is implemented in a portal pane through the use of the **tc:report** tag or the **tc:reportLink** tag. Details about these tags are found in Using Tags to Embed Reports in Blocks.	Display report results in one of the following ways: • Report results appear directly in the portal pane. • A link in the portal pane allows the user to execute a report, or jump to a specific folder of reports.
Text	Serves both of the following purposes: • Links to a network location. For example: `<a href="networkLocation" target="_blank">networkLocationLabel</a>` ***Important:*** *You must specify a target attribute. If you do not, the network location opens in the same window as TeamConnect.* • Displays text to users upon viewing a particular home page. Messages can be displayed to the entire organization, or to a specific group of users, depending on who has access to the particular home page on which the portal pane appears.	• Access network locations through hyperlinks. • Display messages in plain text or HTML format. HTML content can also include references to files hosted on outside servers.

	The portal pane titled "ClientGuard Messages" in the Administrator End-User Home Page Components image on page 306 is an example of a portal pane displaying text. For more details, see Adding Text.	
Web URL	Links to Web sites outside of TeamConnect. When a user clicks a Web URL link in a portal pane, the specified URL is passed directly to the **Address** field of a new browser window. Web URL links can be set based on what is necessary for the users to access and view. The portal pane titled "Important Web Links" in the Administrator End-User Home Page Components image is an example of a Web URL portal pane. For more details, see Adding Web URLs.	Access Web sites through hyperlinks.
Custom Content	Displays customized content and formatting through use of Java class files. The portal pane titled "My Stocks" in the Administrator End-User Home Page Components image is an example of a Custom Content portal pane. ***Note:*** *To create a Custom Content portal pane, you need extensive experience with the Java programming language.* For more details, see Adding Custom Content.	Access specialized content that is created with Java class files, such as TeamConnect publications, weather, stocks, and so on.
WebIntelligence URLs	(WebIntelligence only) Link to WebIntelligence reports. ***Important:*** *You must have System Preferences set appropriately.*	Access WebIntelligence software through hyperlinks.

1.1.8.2.4.1 Adding Action Links

To add an action link to portal pane content

1. Open the appropriate portal pane.
2. In the **Portal Pane Contents** section, select **Action Link** from the **Content Type** drop-down list.
3. Fill in the appropriate fields as described in the Portal Pane Content of Type Action Link table.
4. Click **add more** to add more content. If you are finished adding content, click **Save**.

   The action link is added to the portal pane. You will see your changes when you add the portal pane to a home page.

Portal Pane Content of Type Action Link

Field	Description
Label	Enter a label (maximum 250 characters) that uniquely identifies the action link. The label you enter appears as a link within the user's portal pane. ***Tip:*** *Start your label with a verb that identifies the action that the user can perform. For example, "Create a contact", "Schedule an appointment", or "Generate a document".*
Unique Key	Enter an alphanumeric combination to uniquely identify the portal pane content. After you save the settings, you can edit this field only if it is blank. The unique key cannot contain spaces, underscores, punctuation, or any special characters and cannot exceed 50 characters in length. This is an optional field.
Links to	Select the action that should occur when the user clicks the link from the portal pane. The action will either be to create a new record or to open a tool. ***Caution:*** *If you link to a user tool to which a user is not granted rights, the hyperlink still appears in the user's portal pane. However, the user will not be able to access the tool.*
Custom Object Type	Select the specific custom object definition whose records you would like the user to create from the portal pane.

	This field appears when **New Custom Object** is selected from the **Links to** drop-down list.
Icon Image File	(Optional) Select an appropriate icon to display next to the action link within the portal pane. Icon image files are in the following location: **//Top Level/System/Icons**.
Wizard	(Optional) Select the wizard that the user will access when clicking the action link in the portal pane. The wizards defined for the object selected in the **Links to** field are available for you to select.
Order	Enter an integer to indicate the display order of the action link within the portal pane. If a portal pane contains more than one content item, they will be displayed in the order specified in this field, lowest number first. ***Note:*** *Items with the same display order are sorted and displayed alphabetically.*
Alignment	Select either **Left**, **Center**, or **Right**, to position the action link accordingly in the portal pane.

1.1.8.2.4.2 Adding Object Links

To add a record object link to portal pane content

1. Select **Portal Panes** from the **Go to** list.

   The Portal Panes screen appears.

2. Open the appropriate portal pane.
3. In the **Portal Pane Contents** section, select **Object Link** from the **Content Type** drop-down list.

   The **Object Type** and **Object** fields appear in the **Portal Pane Contents** section.

4. Fill in the appropriate fields as described in the Portal Pane Content of Type Object Link table.
5. Click **add more** to add more content. If you are finished adding content, click **Save**.

   The object link is added to the portal pane. You will see your changes when you add the portal pane to a home page.

The following table describes the object link items in the **Portal Pane Contents** section.

Portal Pane Content of Type Object Link

Field	Description
Label	Enter a label (250 characters max) that uniquely identifies the object link. The label appears as a link within the portal pane.
Unique Key	Enter an alphanumeric combination to uniquely identify the portal pane content. After you save the settings, the unique key appears as read-only text and cannot be deleted or modified. The unique key cannot contain spaces, underscores, punctuation, or any special characters and cannot exceed 50 characters in length. This is an optional field.
Links to	Select the desired object. The appropriate records are filtered in the **Object** field.
Object	Select an existing record from either a search module or a drop-down list. Users will link to the record from the portal pane. The options in the **Object** field vary depending on the item you selected from the Links to drop-down list. • When selecting **Contact**, **Project**, or **Account**, the **Object** field appears as a search module. • When selecting **Group** or **Forum**, the **Object** field appears as a drop-down list. ***Note:*** *Only Global Forums are accessible through portal panes.* • When selecting **User**, the **Object** field can be displayed as either a search module or a drop-down list. This function for **User** is dependent on your System Properties. Selecting **Search View** from the **Links to** drop-down list requires further explanation. For more information, see Adding Search View Object Links.
Order	Enter an integer to indicate the display order of the object link within the portal pane. If a portal pane contains more than one content item, they will be displayed in the order specified in this field, lowest number first. ***Note:*** *Items with the same display order are sorted and displayed alphabetically.*

Alignment	Select either **Left**, **Center**, or **Right**, to position the object link accordingly in the portal pane.

Adding Search View Object Links

A **Search View** object link provides the user with a link to a search view in the portal pane, which is different from displaying the search view itself in the portal pane, as described in Adding Search Views.

Whether the **Filter** or **Results** tab appears when the user clicks a **Search View** object link depends on the search view's settings:

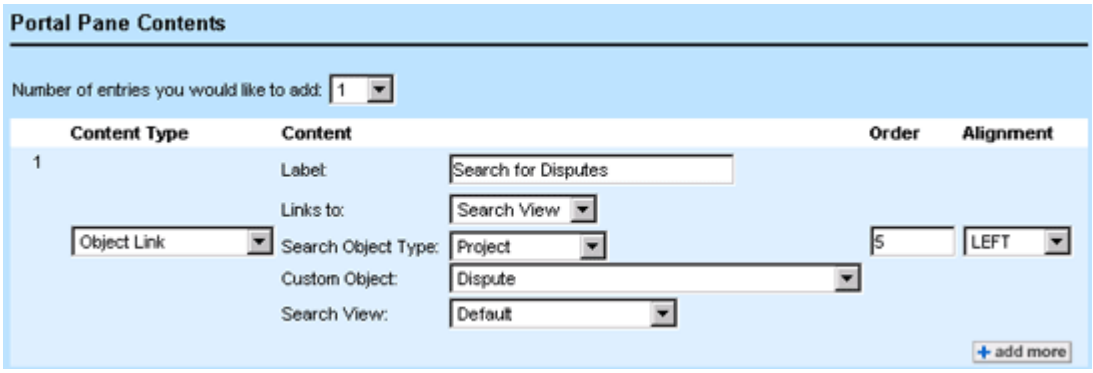
- If the **Auto Search** check-box is selected in the search view, then the **Results** tab appears. This is useful when the search view has search conditions predefined in the **Filter Display** tab.
- If the **Auto Search** check-box is not selected in the search view, then the **Filter** tab appears. The user can then enter the appropriate criteria.

For more details about search view settings, see Creating Search Views.

To add a search view object link to portal pane content

1. Open the appropriate portal pane.
2. In the **Portal Pane Contents** section, select **Object Link** from the **Content Type** drop-down list.
3. Fill in the appropriate fields as described in the Portal Pane Object Link of Type Search View.
4. Click **add more** to add more content. If you are finished adding content, click **Save**.

   The search view object link is added to the portal pane. You will see your changes when you add the portal pane to a home page.



Portal Pane Object Link of Type Search View

The following table describes the object link items of type Search View in the **Portal Pane Contents** section.

Portal Pane Object Link of Type Search View

Field	Description

Label	Enter a label (maximum 250 characters) that uniquely identifies the object link to the search view. The label you enter appears as a link within the user's portal pane.
Unique Key	Enter an alphanumeric combination to uniquely identify the portal pane content. After you save the settings, you can edit this field only if it is blank. The unique key cannot contain spaces, underscores, punctuation, or any special characters and cannot exceed 50 characters in length. This is an optional field.
Links to	Select **Search View** from the list.
Search Object Type	Select the object in which the user will search for records from the portal pane. When selecting **Project**, the **Custom Object** drop-down list appears.
Custom Object	Select the specific custom object for which you would like the user to search. This field appears when **Project** is selected from the **Search Object Type** drop-down list.
Search View	Select the search view that the user will see in the portal pane. The options in this drop-down list display the search views of the object selected in the **Search Object Type** drop-down list, which can be used in the Portal Pane. For more details about search views, see Creating Search Views.
Order	Enter an integer to indicate the display order of the Search View object link within the portal pane. If a portal pane contains more than one content item, they will be displayed in the order specified in this field, lowest number first. ***Note:*** *Items with the same display order are sorted and displayed alphabetically.*
Alignment	Select either **Left**, **Center**, or **Right**, to display the Search View object link accordingly in the portal pane.

1.1.8.2.4.3 Adding Search Views

To add a search view to portal pane content

1. Open the appropriate portal pane view.
2. In the **Portal Pane Contents** section, select **Search View** from the **Content Type** drop-down list.
3. Fill in the appropriate fields as described in the Portal Pane Content of Type Search View table.
4. Click **add more** to add more content. If you are finished adding content, click **Save**.

   The search view is added to the portal pane. You will see your changes when you add the portal pane to a home page.

Portal Pane Content of Type Search View

Field	Description
Label	Enter a label (maximum 250 characters) for the search view results that appear within the user's portal pane. ***Tip:*** *Use the search view name.*
Unique Key	Enter an alphanumeric combination to uniquely identify the portal pane content. After you save the settings, you can edit this field only if it is blank. The unique key cannot contain spaces, underscores, punctuation, or any special characters and cannot exceed 50 characters in length. This is an optional field.
Object Type	Select the type of records for which you would like the user to search from their portal pane. If you select **Project**, the **Custom Object** drop-down list appears.
Custom Object	Select the specific custom object definition for whose records you would like the user to search.
Search View	Select the search view to display in the user's portal pane. This drop-down list includes only search views for the selected object which are specifically set to be available in home pages. For more details, see Creating Search Views for Home Pages.

View Type	Specify how the results of the search view appear in the user's portal pane by selecting either: • **Tabular view**—Search results appear in a table format, like the **Results** tab of a search screen. • **Drop-down list**—Search results appear in a drop-down list. ***Note:*** *In the present version of TeamConnect, regardless of which choice you specify, the tabular view will be used.*
Max Search Results	Enter the maximum number of search results that you want returned to the portal pane. If the search finds more results than the maximum number that you allot, a **More** link appears in the user's portal pane on which they can click to view the additional results. ***Caution:*** *It is recommended that the* ***Maximum Search Results*** *not exceed 1000. Specifying a larger number may affect performance.*
Order	Enter an integer to indicate the display order of the search view within the portal pane. If a portal pane contains more than one content item, they will be displayed in the order specified in this field, lowest number first. ***Note:*** *Items with the same display order are sorted and displayed alphabetically.*
Alignment	Select either **Left**, **Center**, or **Right**, to position the search view accordingly within the portal pane.

1.1.8.2.4.4 Adding Text

To add text to portal pane content

1. Open the appropriate portal pane view.
2. In the **Portal Pane Contents** section, select **Text** from the **Content Type** drop-down list.
3. Fill in the appropriate fields as described in the Portal Pane Content of Type Text image.
4. Click **add more** to add more content. If you are finished adding content, click **Save**.

   The text is added to the portal pane. You will see your changes when you add the portal pane to a home page.

Portal Pane Content of Type Text

Field	Description
Label	(Optional) Enter a label (maximum 250 characters) for the text that appears within the user's portal pane.
Unique Key	Enter an alphanumeric combination to uniquely identify the portal pane content. After you save the settings, you can edit this field only if it is blank. The unique key cannot contain spaces, underscores, punctuation, or any special characters and cannot exceed 50 characters in length. This is an optional field.

Text	Enter plain text or HTML content that you want to display as a message within the user's portal pane (up to 2000 characters).
	Or, enter the following code to create a link to a network location:
	`<a href="Network Location" target="_blank">networkLocationLabel</a>`
	Note: *HTML code in text content can be fairly rich in functionality. For example, the code snippet below establishes a frame, within the portal pane, that acts similarly to a browser, displaying dynamic content:*
	`<iframe>`
	`src ="http://www.google.com" width="100%" height="300" frameborder="0" align="middle" scrolling="yes">`
	`</iframe>`

Order	Enter an integer to indicate the display order of the text. If a portal pane contains more than one content item, they will be displayed in the order specified in this field, lowest number first. ***Note:*** *Items with the same display order are sorted and displayed alphabetically.*
Alignment	Select either **Left**, **Center**, or **Right**, to position the text accordingly within the portal pane.

1.1.8.2.4.5 Adding Web URLs

To add a web URL to portal pane content

1. Open the appropriate portal pane view.
2. In the **Portal Pane Contents** section, select Web URL from the **Content Type** drop-down list.
3. Fill in the appropriate fields as described in the Portal Pane Content of Type Web URL table.
4. Click **add more** to add more content. If you are finished adding content, click **Save**.
5. The web URL is added to the portal pane. You will see your changes when you add the portal pane to a home page.

   An alternate way of embedding a URL in a portal pane is to place HTML code into text content, as described in Adding Text.

Portal Pane Content of Type Web URL

Field	Description
Label	Enter a label (maximum 250 characters) for the Web URL. The label you enter appears as a link within the portal pane.
Unique Key	Enter an alphanumeric combination to uniquely identify the portal pane content. After you save the settings, you can edit this field only if it is blank. The unique key cannot contain spaces, underscores, punctuation, or any special characters and cannot exceed 50 characters in length. This is an optional field.
URL	Enter the address of the Web site to which your users need access. If the URL that you enter does not begin with either http:// or https://, TeamConnect appends http:// to the URL.
Order	Enter an integer to indicate the display order of the Web URL within TeamConnect. If a portal pane contains more than one content item, they will be displayed in the order specified in this field, lowest number first. ***Note:*** *Items with the same display order are sorted and displayed alphabetically.*
Alignment	Select either **Left**, **Center**, or **Right**, to position the text accordingly within the portal pane.

1.1.8.2.4.6 Adding Custom Content

To add custom content to portal pane content

1. Open the appropriate portal pane view.
2. In the **Portal Pane Contents** section, select **Custom Content** from the **Content Type** drop-down list.
3. Fill in the appropriate fields as described in the Custom Portal Pane Content table, and then click **add more**.

   The newly added custom content appears in a list at the bottom of the **Portal Pane Contents** section.
4. Click **Save**.

   The custom content is added to the portal pane. You see your changes when you add the portal pane to a home page.

Custom Portal Pane Content

Field	Description
Label	Enter a label (maximum 250 characters) for the custom content that appears in a user's portal pane.
Unique Key	Enter an alphanumeric combination to uniquely identify the portal pane content. After you save the settings, the unique key appears as read-only text and cannot be deleted or modified. The unique key cannot contain spaces, underscores, punctuation, or any special characters and cannot exceed 50 characters in length. This is an optional field.
Class Name	Enter the name of the Java class file that you would like to generate the HTML within the portal pane. The Java class file you specify must be uploaded to the **Top Level/System/ Portal** folder in the **Documents** area. In the end-user interface, portal panes with custom contents may have the **Edit Parameters** hyperlink, where the users may change the desired parameters through the interface.
Order	Enter an integer to indicate the display order of the custom content. If a portal pane contains more than one content item, they will be displayed in the order specified in this field, lowest number first.

	Note: *Items with the same display order are sorted and displayed alphabetically.*
Alignment	Select either **Left**, **Center**, or **Right**, to display the custom content accordingly within the portal pane.

1.1.8.2.4.7 Adding WebIntelligence URLs

To add a WebIntelligence URL to portal pane content

1. Open the appropriate portal pane view.
2. In the **Portal Pane Contents** section, select WebIntelligence URL from the **Content Type** drop-down list.
3. Fill in the appropriate fields as described in the Portal Pane of Type WebIntelligence URL table.
4. Click **add more** to add more content. If you are finished adding content, click **Save**.

   The WebIntelligence URL is added to the portal pane. You will see your changes when you add the portal pane to a home page.

Portal Pane Content of Type WebIntelligence URL

Field	Description
Label	Enter a label (maximum 250 characters) for the WebIntelligence URL. The label you enter appears as a link within the portal pane.
Unique Key	Enter an alphanumeric combination to uniquely identify the portal pane content. After you save the settings, you can edit this field only if it is blank. The unique key cannot contain spaces, underscores, punctuation, or any special characters and cannot exceed 50 characters in length. This is an optional field.
URL	Enter the address of the WebIntelligence page to which your users need access. ***Note:*** *The method of obtaining the link for a WebIntelligence report depends on the version of WebIntelligence you are using. Please see the Business Objects documentation for more information.*

Order	Enter an integer to indicate the display order of the WebIntelligence URL within TeamConnect. If a portal pane contains more than one content item, they will be displayed in the order specified in this field, lowest number first. ***Note:*** *Items with the same display order are sorted and displayed alphabetically.*
Alignment	Select either **Left**, **Center**, or **Right**, to position the Text accordingly within the portal pane.

1.1.8.2.4.8 Portal Panes w ith Multiple Contents

Depending on the tasks your users perform, you can create portal panes that are multi-functional by adding more than one type of content to the portal pane. By adding multiple content items to a single portal pane, you create a portal pane that functions as a specialized menu of your user's most commonly accessed tasks.

For instance, if you have a group of users that creates contacts, accesses projects, and views a particular Web site on a daily basis, you may want to create a portal pane that accomplishes all of these tasks.

The following image is an example of the General tab of a multi-functional portal pane titled "ClientGuard Menu."

Multi-functional Portal Pane Settings in the Designer Interface

The following image shows how the "ClientGuard Menu" portal pane displays on a user's home page. Notice how the labels of the content items in the Multi-functional Portal Pane Settings in the Designer Interface image appear as links in the portal pane in the following image.

Multi-functional Portal Pane in the End User Interface

Keep in mind the usefulness and functionality of a portal pane. Creating a portal pane with a large number of content items could be confusing to your users. In most situations, it is best to create a few portal panes with a few content items rather than creating one portal pane that does everything.

1.1.8.2.5 Creating Portal Panes

There are two locations from which you can create portal panes:

- **Portal Panes** screen—When you create a portal pane from the Portal Panes screen, it is available to all users and you can add it to any home page.
- **Home Page** screen—When you create a portal pane from the Home Pages screen, it is only available on the specific home page from which you created it. It never appears in the list displayed on the Portal Panes screen. For details, see Creating Portal Panes from a Master Home Page.

To create a portal pane

1. In the **Create a new** drop-down list, select **Portal Pane**.
2. In the **Portal Pane Settings** section, fill in the appropriate portal pane settings as described in the Portal Pane Settings table.
3. Using the Portal Pane Content Types and Their Usage table, identify the content you would like to add.

   Tip: *Do not create a portal pane with too much content. Instead, split the content into multiple portal panes.*

4. In the **Portal Pane Contents** section, in the **Number of entries you would like to add** drop-down list, select the number of entries based on how many additional content items you want to add in this portal pane.
5. Select the **Content Type** from the drop-down list.

   Once you select a content type, an associated group of fields appears.
6. Complete all fields associated with the selected content type as follows:
   - For action link content types, see the Portal Pane Content of Type Action Link table.
   - For object link content types, see the Portal Pane Content of Type Object Link table.
   - For search view content types, see the Portal Pane Content of Type Search View table.
   - For text content types, see the Portal Pane Content of Type Text table.

- For Web URL content types, the Portal Pane Content of Type Web URL.
- For custom content types, the Custom Portal Pane Content table.
- For WebIntelligence content types, the Portal Pane Content of Type WebIntelligence URL table.
- Type the **Order**.
- Select the **Alignment** from the drop-down list.

7. When you are done, click **add more**.
8. If you are finished adding content click **Save**.

   The newly created portal pane is saved and added to the list of existing portal panes in the **Portal Panes** screen.

Portal Panes Screen

To add the new portal pane to a home page, see Creating Home Pages.

1.1.8.3 Home Pages

Designing home pages consists of the following:

- Home Page Settings
- Home Page Content

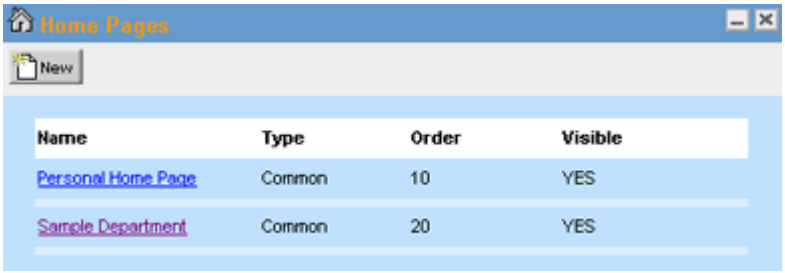
- Creating Home Pages

For a detailed checklist of tasks to complete, see the design checklist.

To access existing home page properties

1. In the **Go to** drop-down list, select **Home Pages**.

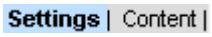
   The **Home Pages** screen appears.



Home Page List Screen

2. Click the hyperlink of the home page you want to edit or to view.

   The corresponding home page screen with the following tabs appears:



   a. **Settings—**Allows you to define general home page settings. For details, see Home Page Settings.

   b. **Content—**Allows you to preview and edit the portal panes of the selected home page or add new portal panes to the selected home page. For details, see Modifying Portal Panes.

1.1.8.3.1 Home Page Settings

You can define settings such as the name and title of a home page on the **Settings** tab of a particular home page. On that tab, you can also define the type of users who are able to view and edit the home page.

Home Page Settings Tab

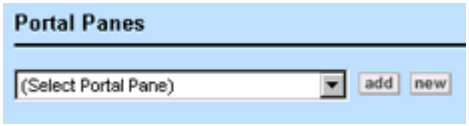
Field	Description
Name	Enter the name (maximum 50 characters) that uniquely identifies the home page. This name appears as a link in the **Home Pages** screen (see the Home Page List Screen image).
Title	Enter the title (maximum 250 characters) that identifies the home page for the user. This title appears in the **Home Pages Menu** (see the Administrator End-User Home Page Components image).

Home Page Type	Select either **Common** or **Group** from the drop-down list to specify which group of users has access to the home page: • **Common** home pages are available to all users. • **Group** home pages are available to a specific group of users. If **Group** is selected, another drop-down list appears in which you must select a specific group of users.
Visible	Select this check-box to make the home page available to the users for which it is defined. If you do not select it, the home page remains unavailable.
Do not allow end users to edit layout or content	Select this check-box to prevent users from changing both the layout and the content. ***Important:*** *While you are designing and creating a home page, this is the only option that should be selected.*
Allow end users to edit layout only	Select this check-box to enable users to change the layout of the home page. Selecting this check-box activates the **Edit Settings** link on the **Personalize Menu** on the home page (see the Administrator End-User Home Page Components image). This function allows users to modify the layout of their own home pages only. Changes made by one user do not affect the home pages of other users.
Allow end users to edit layout & content	Select this check-box to enable users to add content and change the page settings of the home page. Selecting this check-box activates both the **Edit Settings** and **Add Content** links on the **Personalize** menu on the home page (see the Administrator End-User Home Page Components image). This function allows users to modify the layout and content (portal panes) of their own home pages only. Changes made by one user do not affect the home pages of other users. ***Tip:*** *Do NOT select this check-box while you are designing home pages and portal panes for users.*

1.1.8.3.2 Home Page Content

Defining home page content consists of adding portal panes to a home page. You can add existing portal panes, or you can create and add new portal panes directly from the **Content** tab of the **Home**

Pages screen, as shown in the following image. For details, see Adding Portal Panes to Home Pages.



Home Page Content Tab with No Portal Panes Added

The following table explains the initial contents of the **Home Page Content** tab:

Home Page Content Tab with No Portal Panes Added

Field or button	Description
Select Portal Pane	Select the appropriate portal pane to add to the home page from this drop-down list.
add	Click to add the selected portal pane to the displayed home page.
new	Displays the title bar of a new portal pane. You must save the home page before you can edit the new portal pane.

1.1.8.3.3 Creating Home Pages

The following table lists a few tips to keep in mind when creating different home pages for your users:

Tips for Creating Home Pages

Tips	Home page settings
If you would like all users to have a home page in which they can add, delete and rearrange content as they please, create a **Common** home page with the options in the **Home page settings** column selected. Do not add portal panes. This way, users will have a blank home page to work with that they can modify to meet their individual needs. ***Tip:*** *Make sure users are trained on how to add content that is useful to them.*	• Visible • Allow end users to edit layout and content
If you would like users to have a home page in which they can only rearrange the content layout, create a Common or Group home page with the options in the **Home page settings** column selected.	• Visible • Allow end users to edit layout only

If you would like to create a static home page to which only you can make changes but users cannot, create a home page with the options in the **Home page settings** column selected.	• Visible • Do not allow end users to edit layout or content
Create a Common home page that is useful to the entire organization. Add portal panes that are useful for all users.	As deemed necessary.
Create various Group home pages that are useful to specific groups of users. Add only portal panes that accomplish the needs of the specified group members.	As deemed necessary.

To create a home page

1. In the **Create a new** drop-down list, select **Home Page**.

   The **Settings** tab of a new home page appears.

2. Fill in the fields to specify the home page settings, as described in the Home Page Settings Tab table.

3. Click the **Content** tab and add the desired portal panes to the home page as described in the Home Page Content Tab with Portal Panes Added table.

   Copies of the portal panes as they are currently defined are added to the home page. For details, see About Master Home Pages and Portal Pane Templates.

4. Click **Save**.

5. If you added the RSS Feeds Portal Pane, you should set the following content definitions:
   - In the **Go To** drop-down list, select **Portal Panes**.
   - Navigate to the **RSS Feeds** link and select it.
   - From the **General** tab, click the **Edit Parameters** link.
   - From the resulting popup, type the target RSS Feed URL in the URL box.
   - From the **Maximum number of feeds** drop-down list, select the number of feed items to display in the Portal Pane.
   - To sort feed items with most recent items on top, check the **Most recent updates** box.
   - Click **OK**. These changes will apply to the RSS Portal Pane on Home Pages that you assign to multiple users.
   - From the RSS Feeds Portal Pane page, click **Save and Close**.

6. If you are ready for your users to start using the new home page, click the following hyperlinks to activate your new home page:
   - Synchronize Settings

- Synchronize Content

Important: *Do not synchronize settings or content until you are ready for your users to view and start using your new home page.*

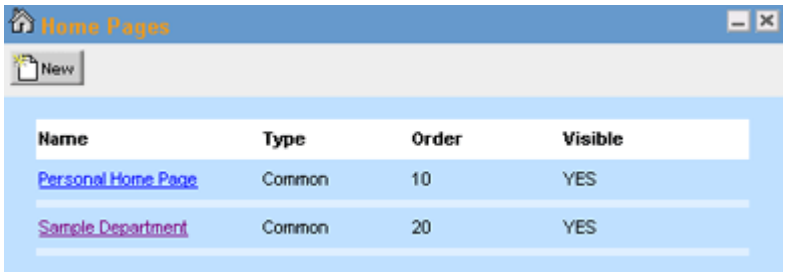
For details about synchronizing settings and content, see Synchronizing Master Pages, Templates, and Copies.

Newly created home pages do not appear in the user's **Home Pages** menu until the user logs off and then logs into TeamConnect again.

To delete a home page

1. In the **Go to** drop-down list, select **Home Pages**.

   The **Home Pages** screen appears.



Home Page List Screen

2. Select the check-box of the home page you would like to delete, and then click **delete**.

   If the home page you selected is visible to a user, it remains visible until the user logs off and logs in to TeamConnect again.

1.1.8.4 Customization Guidelines

Users can customize editable home pages and portal panes to which they have access. When you create new home pages and portal panes from the **Designer** interface, you can set various edit options so that users can customize particular home pages and portal panes to meet their own needs. For details about all home page and portal pane edit options, see Modifying Home Pages and Portal Panes.

New home pages, which you create by selecting **Home Pages** from the **Go to** drop-down list, are master home pages. When users customize their home page, they are actually modifying a copy of the master home page.

The same concept holds true for portal panes. When you create a new portal pane by selecting **Portal Panes** from the **Go to** drop-down list, you create a template. Users who are granted the ability, can modify copies of that portal pane template.

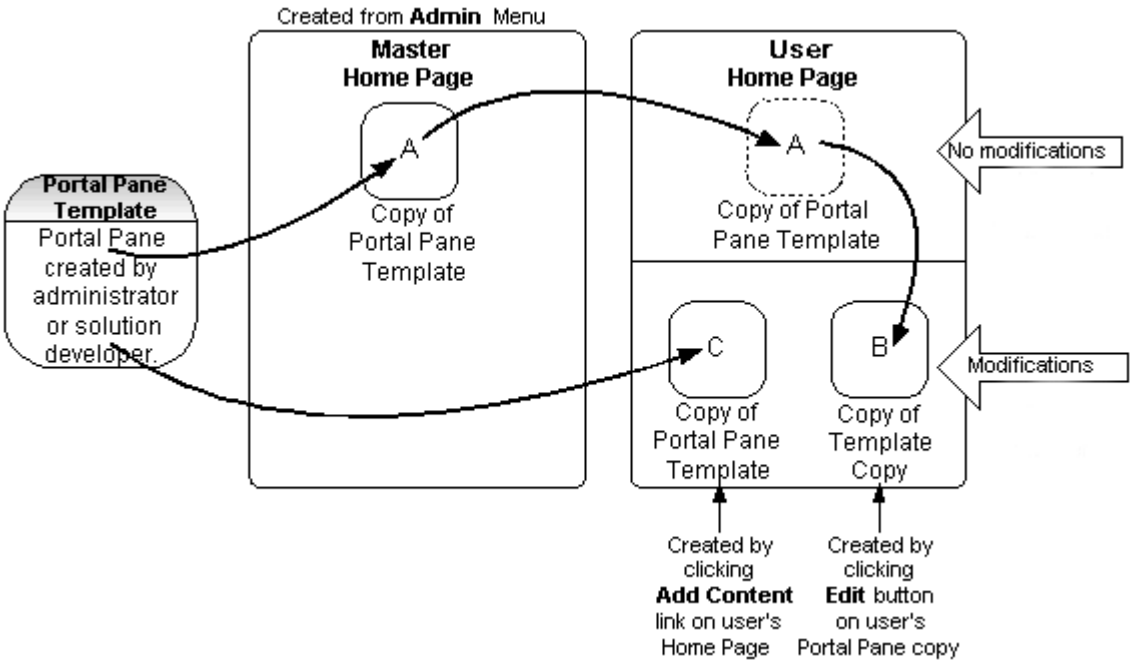
For details about creating and adding home pages and portal panes, see the following sections:

- About Master Home Pages and Portal Pane Templates
- Modifying Home Pages and Portal Panes

- Creating Home Pages
- Creating Portal Panes
- Adding Portal Panes to Home Pages

1.1.8.4.1 About Master Home Pages and Portal Pane Templates

It is important to understand the concepts of master home pages, portal pane templates, and copies. The concepts are best conveyed by the following diagram. It shows what happens to the portal pane template that is the original portal pane, created on the **Portal Panes** screen.



Relations between Master Home Pages, Portal Pane Templates, and Copies

The following table explains details about each of the portal pane copies shown in the diagram.

Explanation of the Diagram

Home page	Portal pane copy	Explanation
Master	**A**	When you add a portal pane template to master home page, you are actually adding a copy of that portal pane template.
User	**A**	When users first view their own home page, they see the same master home page that you created--including the same copy of that portal pane template.
User	**B**	As soon as a user clicks **edit** on a portal pane on his or her own home page, the template copy is copied and replaced with a revised version. In other words, the user has a copy of a copy

		posted on their home page, and portal pane A is no longer visible on their home page. The user only sees his or her customized version of the portal pane.
User	**C**	If a user has the ability to add portal panes, they can click the **Add Content** link on their home page to add a copy of the original portal pane template to their own home page.

In regard to home pages, users initially see an exact replica of the master home page. As soon as users view it, their home page becomes a copy. Therefore, any changes you make as the solution developer, your users cannot see if they have already viewed the home page.

However, there is an exception. If you have deleted a portal pane, then it is deleted from your users' home pages—unless a user has previously modified it. If that is the case, the user's copy of the portal pane that you deleted remains on their home page with their modifications.

Points To Remember

You can synchronize both copies of home pages and portal panes with their master. For details, see Adding Portal Panes to Home Pages.

What users do see is the sum of the master home page plus their own changes to either home pages or portal panes. For this reason, as stated previously, it is important NOT to set the following home page options until after you are finished designing and creating all of your home pages and portal panes:

- **Visible**
- **Allow end users to edit layout only**
- **Allow end users to edit layout & content**

In other words, the only home page setting that should be turned on when you are designing and creating home pages and portal panes is **Do not allow end users to edit layout or content**.

If you do choose one of the options listed in the table above before you are finished creating a home page or portal pane, you can synchronize your master home page and portal pane template changes with the home page and portal pane copies that have already been viewed by other users. For more details, see Adding Portal Panes to Home Pages.

1.1.8.4.2 Modifying Home Pages and Portal Panes

Users can modify their home page and portal panes if given the ability to do so. Solution developer changes and user changes both affect home pages and portal panes. If users modify their home pages and portal panes, the masters, templates, and users' copies will get out-of-sync with each other.

The good news is that as the solution developer, you have total control. If it is necessary at any time to synchronize the content or settings of a home page or portal pane—or both—you can click the **Synchronize Content** and **Synchronize Settings** hyperlinks to do so.

When you synchronize content or settings of home pages or portal panes, you overwrite all modifications that users made to their home pages. The users see the content and settings as they appear in the master home pages and portal panes.

1.1.8.4.2.1 Synchronizing Master Pages, Templates, and Copies

Part of the process of modifying home pages and portal panes is to synchronize your changes from the master home page and template portal panes with the copies that the users have of each.

Synchronizing also plays an important role in creating home pages (not portal panes). To activate a new home page, you must synchronize it. If you are in the process of designing a new home page, it is good practice to NOT synchronize settings or content until your design is final. That way, there is no concern about overwriting user's modifications, in case they decide to customize their home pages or portal panes.

The steps for synchronizing are included within the appropriate instructions in this documentation. It is a matter of clicking links (shown in the following figure), which are in the following locations on the **Designer** interface, at the top of the following screens:

- **Home Pages** screen
- **Portal Panes** screen

Important: *You must save all your changes to the master home page BEFORE you synchronize settings or content.*

Home Page and Portal Pane Synchronize Links

1.1.8.4.2.2 About Modifying Home Pages

Let's say that you give your users the ability to modify home pages by selecting either the **Allow end users to edit layout only** or **Allow end users to edit layout and content** check-box when you create it. When you do NOT click the following synchronize hyperlinks:

- Any changes the user makes to the page settings take precedence over solution developer changes.
- If you, as a solution developer, add portal panes to a home page, the user will see that particular instance of that portal pane. Even if a user clicks the **Add Content** link from the **Personalize** menu to add a portal pane to a home page, and the solution developer adds the same portal pane to the same home page. As a result, the user sees both of those portal panes on their home page. The user can delete the duplicated portal pane, if desired.
- Modifications to home pages do not take effect immediately. Whether adding, deleting or modifying a home page, users must log off, and then log back on to see the changes.

What Happens to Modified Home Pages?

As a TeamConnect solution developer, it is important to be aware of the effects that both user and solution developer changes have on home pages.

You may modify home pages according to the following table:

Effects of Home Page Modifications

Type of user performing task	Task	Changes apply to	Effects of synchronizing
End	Directly from a user's own home page, he or she can click **Add Content** or **Edit Settings** links, and click arrows located on portal panes' title bars to rearrange the layout (if the corresponding options are set appropriately).	• User's home page only.	User's modifications are overwritten with master home page.
Solution developer	From **Go to > Home Pages**, open the appropriate home page. Click the **Portal Panes** tab. Click the arrows located on portal panes' title bars to rearrange the layout.	• Master home page. • Any users who have not yet viewed the master home page.	All copies of this master home page are overwritten.

1.1.8.4.2.3 About Modifying Portal Panes

The same theory that applies to modifying home pages holds true for modifying portal panes. If you give your users the ability to edit or move a portal pane by selecting the **Editable by user** or the **Show title bar** check-box when you create it, any changes the user makes take precedence over solution developer changes.

For example, if a user clicks **edit** on a portal pane and customizes that portal pane, the changes appear only on their home page. Any changes that you made to that same portal pane do not appear on the user's home page. Even if you delete that portal pane from the **Designer** interface, the user still sees their own edited version.

In short, the moment a user customizes a portal pane, those changes take precedence over any solution developer changes to that same portal pane. However, if a user does not edit a portal pane, then solution developer changes take precedence--including the deletion of a portal pane.

As a TeamConnect solution developer, it is important to be aware of the effects that both user and solution developer changes have on portal panes.

You may modify portal panes according to the following table:

Effects of Portal Pane Modifications

Type of user performing task	Task	Changes apply to	Effects of synchronizing

End	Directly from a user's own home page, he or she can click edit in the appropriate portal pane.	User's portal pane copy only.	User's modifications are overwritten with portal pane template.
Solution developer	From **Go to > Portal Panes**, open the appropriate portal pane.	Portal pane template only.	All copies of this portal pane template are overwritten.
Solution developer	From **Go to > Home Pages**, open the appropriate home page, click the **Portal Pane** tab, and then click **edit** on the appropriate portal pane.	Solution developer's copy of portal pane on master home page.	All copies of this portal pane template are overwritten.

The Relations between Master Home Pages, Portal Pane Templates, and Copies diagram shows the different ways in which users can modify editable portal panes, and the results of those modifications (without synchronizing).

To take the task of a user editing a portal pane further—if a user clicks **edit** on a portal pane of their own home page, modifies it, and saves the changes, your actions, as a solution developer, can affect the results. The following table shows the various actions that you might take and the results of those actions. Keep in mind that the results listed are what happens before you click **Synchronize Contents** or **Synchronize Settings** hyperlinks. After you synchronize, portal pane templates overwrite any of its copies.

What Happens when User and Solution Developer Edit Portal Panes

Solution developer and user actions	Results
None.	• The modified portal pane displays only on the user's home page as a separate copy of the portal pane template.
From **Home Pages** > **Portal Panes**, you edit a copy of a portal pane template. User modifies their own copy of that template from their own home page.	• Your changes are reflected on your copy of the portal pane template. (You do not see user's changes). • User's changes are reflected on their own copy of the portal pane template. (User does not see your changes).
From **Portal Panes**, you edit a portal pane template. User modifies their own copy of that template from their own home page.	• Your changes are reflected on the portal pane template only. • Your changes only take effect on users' portal pane copies that were created after the template was updated.

	• User's changes are reflected on their own copy of the portal pane template. (User does not see your template changes).
From **Home Pages** > **Portal Panes**, a copy of a portal pane template. User modifies their own copy of that template from their own home page.	• Portal pane is deleted if it is not editable or has not been modified by the user, yet. • Does not effect portal panes that have already been viewed by a user.

The following table lists modifications to portal panes (of which the content and settings were synchronized) and when those changes take effect:

When Changes Take Effect in Portal Panes

Modification	Effective
Adding portal panes to home pages.	When user clicks **Refresh**.
Deleting portal panes that exist on a home page without prior edits by user.	
Deleting portal panes from the **Home Page** screen with prior edits by user.	When user logs off, and then logs back in.
Modifying portal pane from **Home Page** screen.	Immediately.
Modifying portal pane from the **Portal Pane** screen.	When user logs off, and then logs back in.

For each user who has access to an editable portal pane, a separate instance (or copy) of the portal pane template exists. Users with access to an uneditable portal pane, see the solution developer's copy of that portal pane on their home page. Changes to one portal pane copy do not affect another portal pane copy. Nor do the changes affect the portal pane template.

You can modify the following in a portal pane by changing the following settings in the **Portal Pane Settings** section or **Portal Panes Contents** section on the **General** tab of the **Portal Pane** screen:

- Title
- Color
- User options
- Contents

To modify a portal pane

1. Access the portal pane you would like to modify. For more information see, Opening Portal Panes.
2. Depending on your needs, do any of the following:
   - Change the desired portal pane settings, such as whether it is editable by users.
   - Add additional content to the portal pane using the data entry rows. For details see Portal Pane Contents.
   - Edit or delete existing content items in the existing content list, as desired.
3. Click **Save**.

   Important: *You must save all your changes BEFORE you synchronize settings or content.*

4. Click the **Synchronize Settings** and **Synchronize Content** hyperlinks.

   A message to confirm appears.
5. Click **OK**.

   All user copies of this portal pane are deleted and then replaced with the solution developer's updated copy, not the template.

You can delete portal panes in the following two places:

- **Portal Panes** screen—Not only deletes portal pane from the list of portal panes, but also from master home pages and users' home page to which it was added.
- An open master home page—Deletes portal pane from the open master home page and from all copies of that master home page (after clicking the **Synchronize Layout** and **Synchronize Content** hyperlinks).

To delete a portal pane from the portal pane list

1. In the **Go to** drop-down list, select **Portal Panes**.

   The **Portal Panes** screen appears with all existing portal panes.

Portal Panes Screen

2. In the **Portal Panes** list, select the check-box of the appropriate portal pane, and then click **delete**.

   A message to confirm the deletion of this portal pane appears.

3. Click **OK**.

   The portal pane is deleted from the system and you can no longer add it to a home page. For logged in users, this portal pane is deleted from their home page upon clicking **Refresh**, or upon logging out and then logging back in to TeamConnect--even if they have at one time or another modified it.

To delete a portal pane from a master home page

1. In the **Go to** drop-down list, select **Home Pages**.
2. Open the appropriate home page.
3. Select the **Portal Panes** tab.
4. On the title bar of the portal pane you want to delete, click the **Remove Pane** button.

   A message to confirm the deletion of this portal pane appears.

5. Click **OK**.

   The portal pane is deleted.

6. Click **Save**.

   Important: *You must save all your changes BEFORE you synchronize settings or content.*

7. Click the **Synchronize Content** hyperlink.

   A message to confirm that you want to synchronize the master home page with user's copies of this home page appears.

8. Click **OK**.

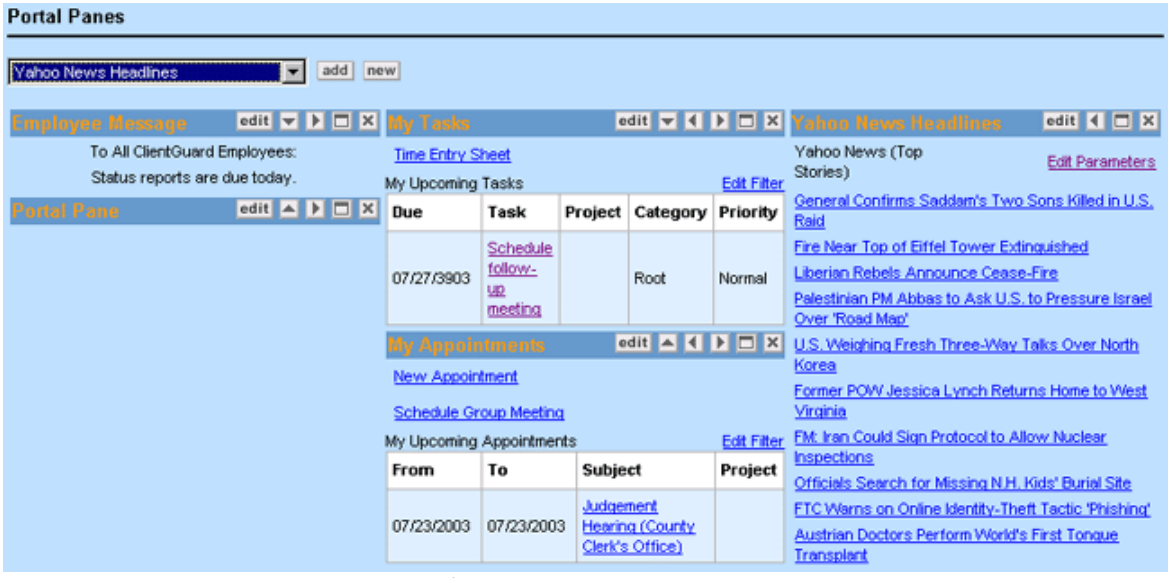
   All home pages are in synchronization with the master home page.

1.1.8.4.3 Modifying Master Home Pages

You can preview the layout of a master home page while you are modifying it.

You can view and edit existing portal panes that are already displayed on a particular home page. You can also add other existing portal panes from the **Content** tab of an existing home page. Further, you can create new portal panes directly from the master home page. For details, see Creating Portal Panes from a Master Home Page.

After clicking the **Content** tab, a layout preview of the selected home page appears, along with the **Select Portal Pane** drop-down list. All of the portal panes that appear on the **Content** tab function exactly as they would on the user's home page. You can modify the function, position, and size of these portal panes by clicking the buttons that appear in the **Title Bar** of each individual portal pane. These buttons will always be visible to you on the **Content** tab, even if they are not visible to the user on the user home page.



Home Page Content Tab with Portal Panes Added

Home Page Content Tab with Portal Panes Added

Item	Description

Select Portal Pane drop-down list	Select the desired portal pane you would like to add to the selected home page. All portal panes that were created from **Portal Panes** are listed.
add	Click to add the selected portal pane to the home page. If you are adding portal panes to a home page that is already available to a user, the additions are not displayed for the user until he or she clicks the **Refresh** hyperlink.
new	Click to create a new portal pane directly from the displayed master home page. ***Important:*** *When you create a new portal pane by clicking this button, the portal pane that you create is not listed on the* ***Portal Panes*** *screen. It is created for this particular home page only.* For details, see Creating Portal Panes from a Master Home Page.
Selected Portal Panes	These are the portal panes that have already been added to the selected home page. The way that they are arranged is the way the user will see them.
Title Bars	Click the appropriate buttons that appear on the title bar to modify the individual portal pane accordingly.
edit	Click to change the selected portal pane. Clicking this button opens the **General** tab of the selected portal pane where you can modify the settings of that portal pane. To learn more about modifying portal panes, see About Modifying Portal Panes.
Move Pane	Click the appropriate arrow button to move the selected portal pane to a different position on the home page. Depending on where the portal pane is positioned in the home page, you may have multiple arrow buttons in the title bar that point up, down, left, or right.
Maximize **Minimize**	Click to maximize the selected portal pane. After you click the **Maximize** button, the portal pane occupies the entire width of the screen, but the height of the portal pane remains the same. Conversely, if the portal pane is already maximized, the **Minimize** button is available. Click this button to resize the portal pane to its original size.

Remove Pane	Click the **Remove Pane** button to remove the selected portal pane from the home page. The portal pane is removed after the user clicks the **Refresh** hyperlink in the **Personalize** menu.

1.1.8.4.3.1 Adding Portal Panes to Home Pages

Whether you are adding portal panes to a user's home page or editing portal panes that already exist in a user's home page remember the following:

- The user is unable to view the portal panes that you add until after they refresh their screen.
- Portal panes that you remove are still viewable by the user until after they refresh their screen.
- Changes made to a portal pane from the **Portal Panes** tab of the **Home Page** screen will take effect only for that specific portal pane within that specific home page. Other portal panes are not affected.
- When making changes to a portal pane template that already exists as a copy on a user's home page, the changes you make do not take effect until you synchronize the content and settings. For details, see When Changes Take Effect.
- If you change a portal pane that a user has previously customized, the user's changes take precedence. To learn more about solution developer and user customization, see Modifying Home Pages and Portal Panes.

After you save this new portal pane, you cannot reuse it. The portal pane that you created is only intended for this master home page and is not listed on the **Portal Panes** screen.

To add portal panes to a home page

1. From the **Content** tab of the desired home page, select the appropriate portal pane from the **Select Portal Pane** drop-down list, and then click **add**.
2. Repeat the previous step to add additional portal panes to the desired home page, if necessary.
3. Click **Save**.

   Important: *You must save all your changes BEFORE you synchronize settings or content.*

4. Click the **Synchronize Content** hyperlink.

   A message to confirm that you want to synchronize the master home page with user's copies of this home page appears.

5. Click **OK**.
6. Click the **Synchronize Settings** hyperlink.

   A message to confirm that you want to synchronize the master home page with user's copies of this home page appears.

7. Click **OK**.

   The desired portal panes are added to the master home page and all copies are in sync with it.

1.1.8.4.3.2 Creating Portal Panes from a Master Home Page

To create a portal pane from a master home page

1. In the **Go to** drop-down list, select **Home Pages**.
2. On the **Home Pages** screen, select the home page to which you want to add a portal pane.
3. Click the **Content** tab, and then click **new**.

   A title bar of the new portal pane appears in the open home page.
4. Click **Save** at the top of the **Home Page** screen.

   If you attempt to edit your new portal pane before you save your modified home page, a message appears, which prompts you to do so.
5. Click **edit** on the title bar of the new portal pane.

   The **Portal Pane** screen appears.
6. Enter appropriate portal pane settings as described in the Portal Pane Settings table.

   Tip: *Do not select the **Editable by user** check-box until your new portal pane is finalized and exactly was you want it.*

7. Select the **Number of entries you would like to add** from the drop-down list, based on how many additional content items you want to add to the portal panes.
8. For each data entry row, do the following actions:
   a. Select the **Content Type** from the drop-down list.
   b. Fill in the fields that display beneath the **Content** column.

      For details about each field, see the appropriate content type in the **Portal Pane Contents** section:

      - For action link content types, see the Portal Pane Content of Type Action Link table.
      - For object link content types, see the Portal Pane Content of Type Object Link table.
      - For search view content types, see the Portal Pane Content of Type Search View table.
      - For text content types, see the Portal Pane Content of Type Text table.
      - For web URL content types, see the Portal Pane Content of Type Web URL table.
      - For custom content types, see the Custom Portal Pane Content table.

- For WebIntelligence content types, the Portal Pane Content of Type WebIntelligence URL table.

Tip: *Do not create a portal pane with too much content. Instead, split the content into multiple portal panes.*

c. Type the **Order**.

d. Select the **Alignment** from the drop-down list.

9. Click **add more** to add more content, or click **Save** if you are finished.

Important: *You must save all your changes to the master home page BEFORE you synchronize settings or content.*

10. Click the **Synchronize Content** hyperlink.

    A message to confirm that you want to synchronize the master home page with user's copies of this home page appears.

11. Click **OK**.

12. Click the **Synchronize Settings** hyperlink.

    A message to confirm that you want to synchronize the master home page with user's copies of this home page appears.

13. Click **OK**.

    The newly created portal pane is saved and added to the displayed home page.

1.1.9 Using Global Navigation

Global Navigation determines which tabs appear in the TeamConnect tab bar and which navigation items appear in the sub-tabs when you click on a tab. By making changes in the **Global Navigation** screen, you may reorganize existing navigation items. Navigation items link to these kinds of pages:

- System objects
- Custom objects
- Custom tools
- Special collections related to workflow
- Home pages

When you use Global Navigation to alter the appearance or organization of the tab bar, your changes take effect for each user upon logging in. For you to see your own changes in the end-user interface, you must log out and log back in.

Note: *You cannot alter the appearance and contents of some tabs (**Admin**, **All Services**). TeamConnect controls the contents and positions of these tabs automatically.*

You can reach the Global Navigation portion of the Designer by clicking the **Setup** link in the main user interface, then choosing **Global Navigation** from the **Go To** dropdown list in Designer. Your user group must have Setup rights in order to see the Setup link.

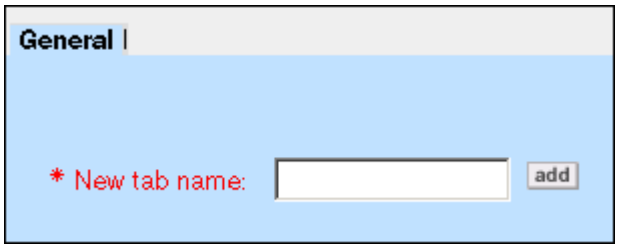
1.1.9.1 Working with Tab Bar Appearance

Each tab represents a high-level grouping of main pages. You may wish to alter the appearance of the tool bar to show more or fewer tabs, or change the positions of tabs. The description here explains how to do those tasks.

To learn how to change the navigation items associated with a specific tab, see Working with Tab Bar Contents.

To create a new tab

1. Type the name of the new tab in the **New tab name** field.
2. Click **Add**.
3. Your new tab appears to the right of all existing tabs. In the end-user interface, it will be to the right of all tabs except Admin and All Services.
4. Click **Save** or **Save and Close**.



Creating a New Tab

To reposition tabs

1. Click on the tab that you wish to reposition.
2. Click on the left-arrow button to move the tab more to the left, or the right-arrow button to move the tab more to the right.
3. If more than one tab needs repositioning, repeat steps 1 and 2 for each tab.
4. Click **Save** or **Save and Close**.

Controls for Repositioning, Deleting, and Renaming Tabs

To delete a tab

1. Click on the tab that you wish to delete.

2. Click on the delete ("**X**") button to delete the tab.
3. In the confirmation message pop-up window, click **Yes**. The remaining tabs adjust to the fill in the gap.
4. Click **Save** or **Save and Close**.

Important: *If you delete a tab, end users do not have any way of accessing the navigation items that were associated with that tab, except by finding them in the All Services tab or if you associate those navigation items with another tab.*

To rename a tab

1. Click on the tab that you wish to rename.
2. Click on **Rename**.
3. A pop-up window asks you to for the new name of the tab. Type the name and press **OK** to close the pop-up window.
4. Click **Save** or **Save and Close**.

1.1.9.2 Working with Tab Bar Contents

Each tab represents a high-level grouping of navigation items. Some tabs, such as Contacts, are associated with only a single navigation item. TeamConnect recognizes when there is only a single item within a tab. In this situation, when an end user clicks that tab, she goes directly to the linked page.

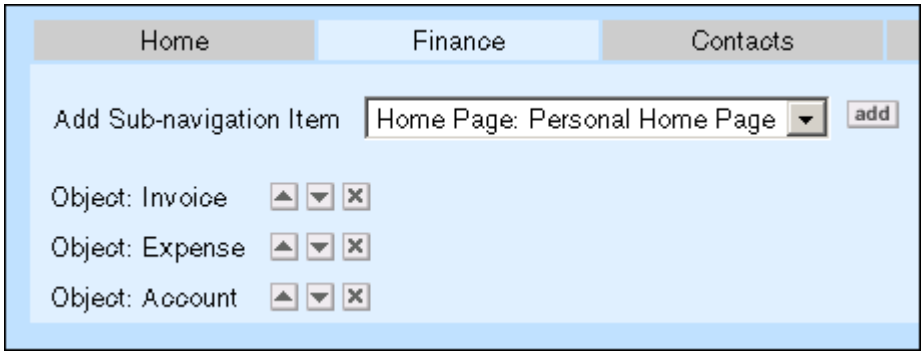
Other tabs, such as Finance, are associated with multiple navigation items. TeamConnect cannot know in advance which item the end user is interested in, so when he clicks such a tab, he sees the available navigation items displayed as sub-tab links in the lower portion of the tab bar. Clicking on one of these sub-tab links then takes him to the linked page.

Only these two forms of navigation are available. You cannot add a navigation item, then add another navigation item as a child of the first item. All items are children of the tab itself.

You can change the navigation items that are associated with a tab in several ways.

To add a new navigation item to a tab

1. Click on the tab that you wish to alter.
2. Click on the **Add Sub-navigation** item drop-down list and choose a navigation item. The types of items available are system objects, custom objects, home pages, and custom tools.
3. Click **Add**.
4. Click **Save** or **Save and Close**.



Controls for Repositioning, Deleting, and Creating Navigation Items

To reposition navigation items

1. Click on the tab that you wish to alter. The existing navigation items are arranged vertically. The topmost item corresponds to the leftmost link in appearance in the lower part of the tab bar; the bottom item in the list is the rightmost link in the tab bar.
2. Click on the up-arrow button to move the item higher, or the down-arrow button to move the item lower.
3. If more than one item needs repositioning, repeat steps 1 and 2 for each item.
4. Click **Save** or **Save and Close**.

To delete navigation items

1. Click on the tab that you wish to alter. The existing navigation items are arranged vertically. The topmost item corresponds to the leftmost link in appearance in the bottom of the tab bar; the bottom item is the rightmost link in the tab bar.
2. Click the delete ("X") button next to an item name.
3. In the confirmation message pop-up window, click **Yes**.
4. Click **Save** or **Save and Close**.

1.1.10 Using Templates

Templates define what information is automatically filled in for each type of record and which sub-object or related records are automatically created. You may use templates with wizards, pre-population rules, and custom action rules.

All of these elements support the following functionality:

- Creating a new record and populating its fields
- Creating related and sub-object records for the current object and populating their fields
- Populating assignees and attendees and specifying the main assignee by role
- Specifying the default category with a static value

- Populating categories
- Specifying a phase for custom objects

Pre-population and custom action rules support the following additional template functionality:

- Populating fields of existing records
- Performing phase changes

For information about using templates with rules, see Using Rules. For information about using templates with wizards, see Whether to Use Templates.

1.1.10.1 Planning Templates

Your template design depends on your organization's business requirements, as well as the object definition in question. To create useful and effective templates, you must do the following actions:

- Complete the object definitions for which you want to create a template. For more information, see Creating and Defining Objects.
- Familiarize yourself with the TeamConnect object model, whose attributes are described in the Object Model: Read This First plus the additional related reference tables.
- Learn how to use Object Navigator, which you may use to define fields in templates. See Using Object Navigator.
- If you need to use templates with wizards, plan any necessary wizards before creating templates. See Creating Wizards.

General Considerations

The following checklist helps you develop a general outline of the templates you may need to create and what kind of information has to be collected:

- Determine the possible scenarios that may take place when a record is created in your organization, such whether users create most records using wizards or also with new, blank records.
- Determine how many wizards and templates you need to create based on the various scenarios, such as one complex wizard with multiple pages and page transition rules creating multiple related records, or multiple small wizards, each taking care of a specific object record.
- Identify the information that must be automatically populated in every new record.
- Determine whether any related records need to be automatically created together with every new record. For example, tasks to be completed, an audit history record to capture the details of a new record, and so on.
- Determine whether any sub-objects need to be added to a record. For example, default categories to add certain blocks, assignees, and so on.
- Identify any fields that may be automatically prepopulated with static or dynamic values in every new record based on naming conventions, company procedures, and other known patterns.

Whether you are defining template fields, sub-objects, or related objects, make sure you understand the following subjects:

- Field Value Types
- Values, attributes, and using Object Navigator (see Using Object Navigator)
- TeamConnect attributes (see the Object Model: Read This First plus the additional reference tables related to this reference).
- Operator Options for Different Attribute Types
- Defining Values in Templates
- Sub-Objects
- Related Objects

Points To Remember

Keep the following points in mind when planning templates:

- Creating templates is optional, except for pre-population rules.
- Some fields, such as descriptions, dates, number fields, and so on, may be prepopulated using simple actions in a wizard--without the use of a template.
- Whether you create templates for use with wizards, for use with rules, or both, the methods of defining which fields to populate and records to create are the same.
- You may create several templates for the same object definition.
- You may use only one template per wizard or rule, yet you may link related object templates to the main object template.
- Once you create a template, you may select it from the **Action** tab of your pre-population or custom action rule screen.

  If you are creating custom action rules for the **User** or the **Group** object definition, you cannot select a template from the **Action** tab.
- You may select existing templates from the **General** tab of wizard screens.
- You may access or create templates on the **Templates** tab of the object definition. However, the following objects do not have a **Templates** tab:
  - Contact Group (Address Book)
  - Document
  - Embedded objects
  - Group
  - Line Items
  - User
- You may create templates for related objects and select them from the main object template. That way, you may reuse the related object template for multiple main objects and scenarios.

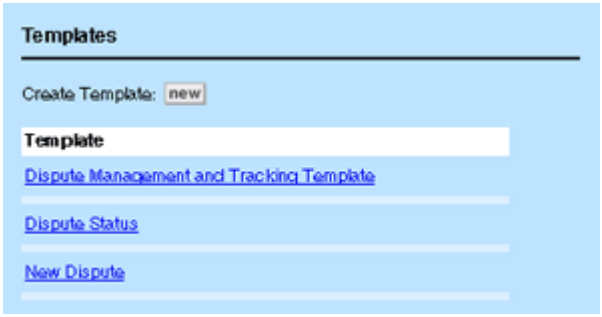
1.1.10.2 Using Template Features

Templates are object-specific, so you may access them from the **Templates** tab of the corresponding object definition.

To open a template screen

1. In the Designer window, from the **Go to** drop-down list select **Object Definitions**.
2. Select the appropriate object definition.
3. Select the **Templates** tab.

   The corresponding **Templates** tab appears with a list of any existing templates defined for the selected object, as shown in the following image.



Templates Tab

4. On the **Templates** tab, do one of the following actions:
   - To open an existing template, click its link.
   - To create a new template, click new.

   The **General** tab of the corresponding template screen appears.
5. Click the **Records** tab to view or modify most template definitions.

The template screen has the following tabs:

General | Records

- **General**—Displays the name of the template and the object it is defined for, and allows you to select a default Documents folder to be associated with each object record. See General Template Information.
- **Records**—Allows you to select fields, sub-objects, and related objects from a tree structure in order to define template fields, their values, and order.

1.1.10.2.1 General Template Information

The **General** tab in the **Templates** screen displays the template's name, its unique key, the name of the object for which it is defined, and the document folder associated with records created by the template.

You may access this tab by following the instructions in Using Template Features.

General Tab on Template Screens

The following table explains the contents of the **General** tab on the **Template** screen:

General Tab on Template Screens

Field	Description
Name	Enter a name that uniquely identifies the template.
Unique Key	Enter an alphanumeric combination to uniquely identify the template within the object definition. ***Note:*** *Templates belonging to different object definitions may have the same key.* The unique key must be 50 or fewer characters in length and may not contain spaces, underscores, commas, punctuation marks, or any special characters. After you save the template, the unique code appears as read-only text and you may not change it.
Default Document Folder	Select a template folder to have its contents automatically added to the **Documents** tab of each record that is created based on the template. If you do not create any template folders, this field displays **(No Item Selected)** in read-only text. For more information about adding folders to records, see Defining Document Folders for Templates.

1.1.10.2.2 Defining Document Folders for Templates

You may select a template folder to have its contents automatically added to the **Documents** tab of each record that is created based on the template. The folder itself is not copied, but any of the following content types may be:

- Subfolders and their contents
- Hyperlinks
- Files, such as Word documents

- Shortcuts to other folders and documents

You may create the appropriate folder structure for the object's templates in the following folder in TeamConnect's Documents area:

Top Level/System/Object Definitions/*objectName*/Template Folders/

Where:

objectName is the name of your template's object definition.

For example, if you create the Negotiations/Offers folders within **Template Folders**, the **Negotiations** folder becomes available on the **General** tab of your template.

Each dispute record that is created based on the template has an **Offers** folder on its **Documents** tab that contains any files in the **Offers** folder, such as the **OfferLetter.doc** file. The **Negotiations** folder is not copied to the record.

To add folders and their contents to templates

1. From the end-user interface, select the **Documents** tab.

   Your personal user folder opens.

2. On the Location Indicator, click **Top Level** and navigate to the following folder:

   Top Level/System/Object Definitions/*objectName*/Template Folders/

   Where objectName is the name of your template's object definition.

3. Click **Create New Folder** and enter a folder name in the **Name** field.
4. Add any subfolders, files, hyperlinks or other contents to the folder.
5. On the **General** tab of the appropriate template screen, select the newly created folder from the **Default Document Folder** drop-down list.

   If no template folders were created for the template's object definition, the **Default Document Folder** field displays **(No Item Selected)** in read only text.

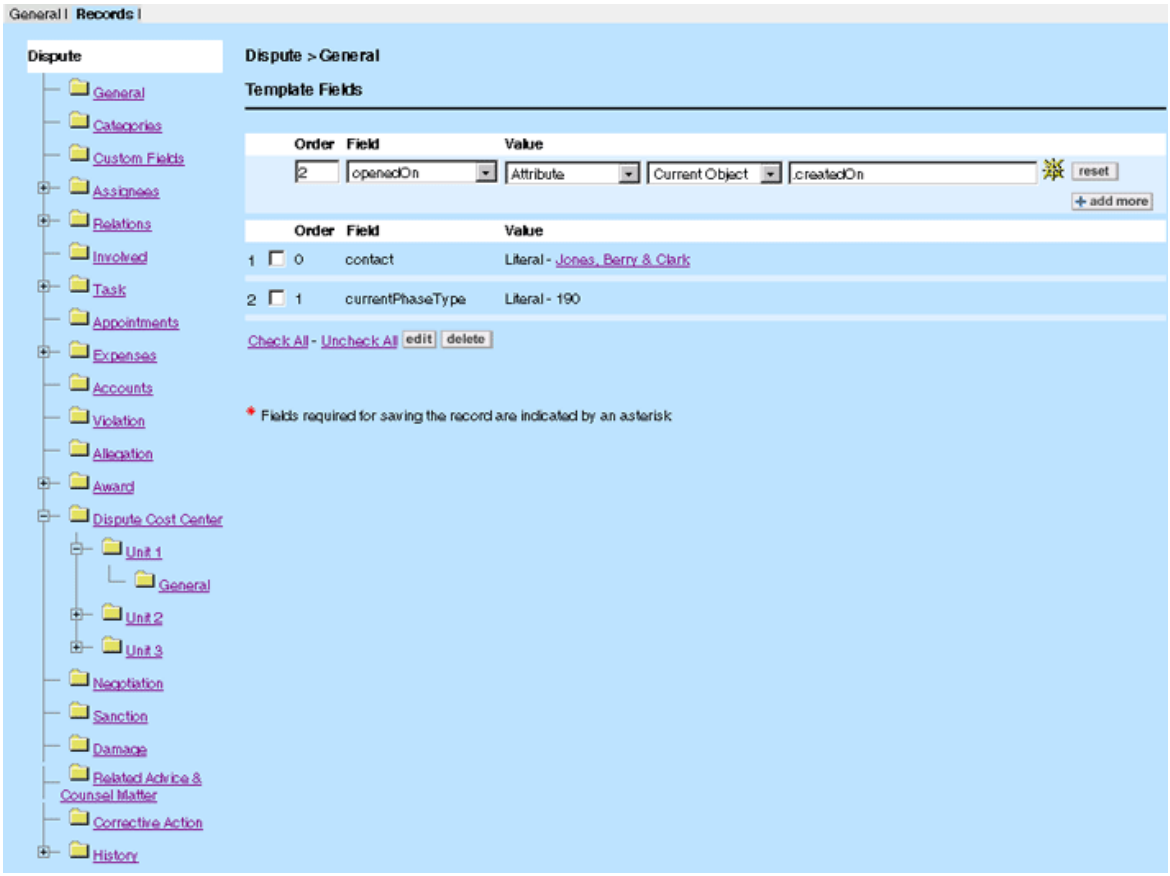
6. Click **Save**.

   The contents of the selected folder are automatically added to the **Documents** tab of each record that is created based on the template.

1.1.10.2.3 Records Tab Structure

The **Records** tab provides a left pane with a directory-tree structure of custom fields, sub-objects, and related objects and a right pane where you may define items in the selected folder.

Depending on the selected folder in the left pane, you may create subfolders for multiple definitions. For example, templates may create multiple involved party records with different categories, if you define multiple subfolders.



Custom Object Records Tab Example

For custom objects, the left pane of the **Records** tab resembles the tree structure of the object definition list that you access in Designer. Every embedded or related child object displayed in the object definition list is available in the **Records** tab. In addition, any applicable sub-objects have folders, such as **Assignees**, **Involved**, **Relations**, **Attendee**, and so on.

When you open an object's folder, you must enter a name in its **Friendly Name** field before you may define any of its values. TeamConnect creates a subfolder with the name you enter, in which it creates a **General** subfolder.

1.1.10.3 Creating Templates

You may access or create templates on the **Templates** tab of the corresponding object definition.

To create a template, you must have the following information:

- Object for which you want to create a template.
- System and custom fields you want the template to automatically populate with default values, such as **Name**, **Opened On**. For more details on fields, see Adding Fields to Templates.
- Sub-objects you want the template to automatically add and populate with default values, such as **Categories**, **Assignees**, and **Relations**. For more details on sub-objects, see Adding Sub-objects to Templates.

- Related objects you want the template to automatically create, such as **Tasks**, **Accounts**, **History**, and **Appointments**. For more details on related objects, see Adding Related Objects to Templates.

When you have all this information available you are ready to create your template.

To create a template

1. Select **Object Definitions** in the Designer window, from the **Go to** drop-down list.
2. Select the appropriate object definition.
3. Select the **Templates** tab.
4. Click **new**.

   The **General** tab of a new, blank template screen appears.
5. In the **Name** field, type a name to identify the template.
6. In the **Unique Key** field, type a name or value that uniquely identifies the template within the object definition.

   A template's unique key may not contain spaces, underscores, punctuation marks, or any special characters.
7. In the **Default Document Folder** drop-down list, select the folder with contents that you want to automatically populate the **Documents** tab of each record created from the template.
8. Click **Save**.
9. Click the **Records** tab.

   The objectName > General folder of the Records tab appears by default.
10. Depending on your template design, add and define the appropriate elements, depending on the following conditions:
    - If you need to define system fields in the template, click the **General** folder (if it is not already displayed).

      See Adding Fields to Templates.
    - If you need to add categories to the template, click the Categories folder displayed on the **Records** tab.

      See Adding Categories to Templates.
    - If you need to add sub-objects to the template, click the corresponding folder displayed on the **Records** tab.

      See Adding Sub-objects to Templates.
    - If you need to add related objects to the template, click the corresponding folder displayed on the **Records** tab.

      See Adding Related Objects to Templates.
11. Click **Save**.

12. Link the template to the appropriate wizard, and when the wizard design is complete, test them both.

1.1.10.3.1 Defining Values in Templates

The process of defining template fields, sub-objects, and related objects essentially consists of assigning certain default values to their respective attributes. These attributes usually represent the actual fields in the end-user interface. The values you specify in your template automatically appear in the corresponding fields in the user interface when the user creates a new record based on your template.

Templates use three basic types of field values: Literal (or "static"), Attribute, and Formula, which may be a combination of Literal, Attribute, and operator values. For more information on field values, see Field Value Types. For the explanation of different operators, see the Operator Options for Different Attribute Types table.

To define fields in a template

1. Make a list of all fields that you want to be automatically prepopulated with the help of the template you are creating.
2. Decide which default values to assign to each field.
3. Make sure no field definitions in the template conflict with the settings in the selected object definition, for example, Names for custom objects.

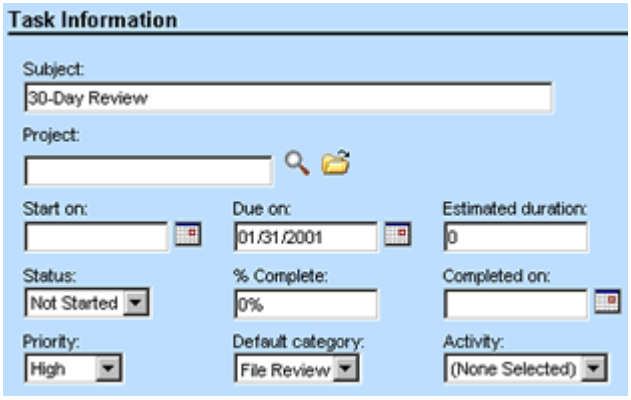
   For more details, see Points To Remember.
4. Decide which fields also require that their corresponding sub-objects must be defined and prepopulated first, for example, Categories or Assignees.
5. Open the Records tab of the appropriate template.
6. Depending on the value type you want to assign to each field, proceed to the following instructions:
   - Defining Fields with Literal Values
   - Defining Fields with Attribute Values
   - Defining Fields with Formula Values

1.1.10.3.2 Adding Fields to Templates

Templates allow you to define certain fields in an object record to be automatically prepopulated with specified values in each new record created through any wizards or rules to which the template is linked.

Prepopulated fields may include record names, descriptions, amounts, dates, default categories, and so on. For example, every time a user creates a task based on a 30-Day Review Task template, you may want the following fields to be automatically populated with specified values such as:

- Subject
- Due on
- Priority
- Default Category



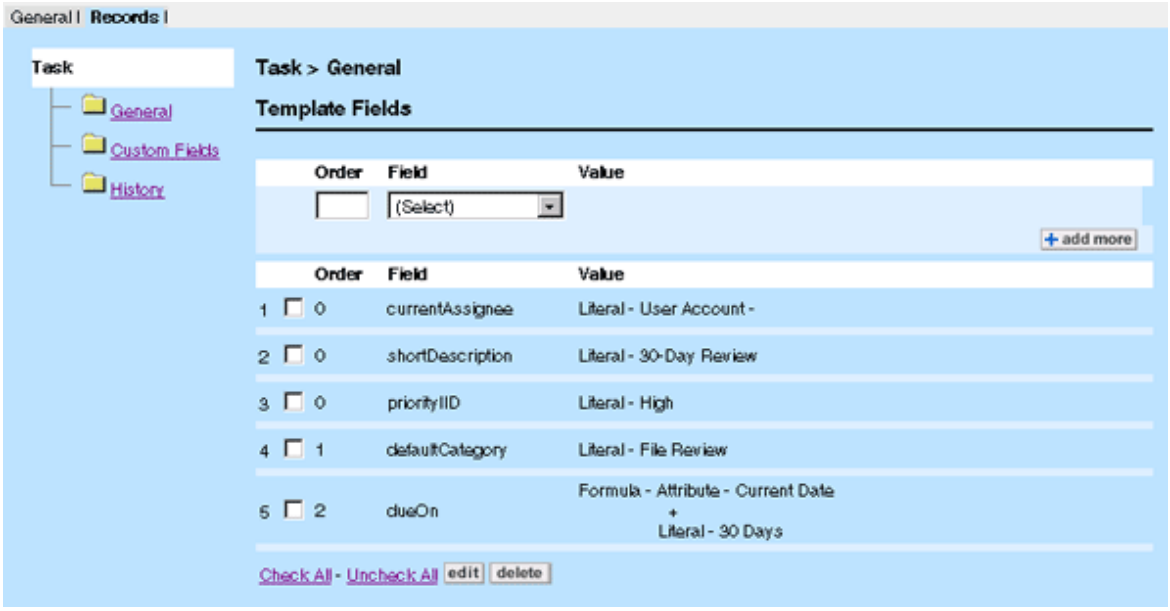
Task Fields in End-user Interface

To accomplish this kind of action using a template, define the following default values to be populated in each record:

Sample Template Values

Field label in the user interface	Field	Value	Default value inserted
Subject	shortDescription	Literal > 30-Day Review	**30-Day Review**
Due on	dueOn	Formula > Attribute > Current Date + Literal > 30 > Days	Depends on the date of creation.
Priority	priorityIID	Literal > High	**High**
Default Category	defaultCategory	Literal > File Review	**File Review**

The following image illustrates the values in the Sample Template Values table.



Sample Task Template Values

You may define fields on the **Records** tab of the template screen in the object definition. For instructions on this operation, see Adding Fields to Templates.

For details on attributes, field value types, and paths, see Field Value Types. For details on operators, see the Operator Options for Different Attribute Types table.

Points To Remember

Keep the following points in mind when adding fields to templates:

- There are certain fields, such as **Default Category** or **Main Assignee**, that you populate only if the corresponding category or assignee is added to the record. This means that in your template, you have to define the corresponding sub-objects first (in this case, a category and an assignee), before setting the corresponding default fields. See Adding Sub-objects to Templates.
- When defining fields for custom objects, keep in mind that the **name** and **number** attributes may or may not be available for selection, depending on the properties defined in the object definition, under the following conditions:
    - If the project name or number are set to be manually entered by the user in the **Name** and **Unique ID** tabs of the corresponding object definition, the corresponding attributes are available in templates.
    - If the project name or number are set to be automatically generated based on the selected pattern in the **Name** and **Unique ID** tabs of the corresponding object definition, the corresponding attributes are not available in templates.
- Depending on your wizard or rule design, all values assigned to the fields of the main record and any of its sub-objects and related objects may be overwritten by your specified wizard or rule actions.

1.1.10.3.2.1 Field Value Types

Regardless of whether you are defining template fields, sub-objects, or related objects, you must specify a field value that is automatically populated when the user creates a new record. As shown in the following table, these field values may be:

- Literal: static
- Attribute: dynamic
- Formula: static-dynamic

Field Value Types

Literal	Attribute	Formula
Values that users manually enter or select, also known as "static" values. For example, the value for the **Default Category** in a claim record is manually set to **Category 1**, which is set to **Auto**.	Values that are based on a value entered or selected in another field and defined using Object Navigator, also known as "dynamic" values. For example, the value for the sub-object **Assignee 2** is set to the user who created the record.	Values that are derived from a combination of static and dynamic values, also known as "static-dynamic" values. For example, the value for the **Due On** date of the related object **Task 1** is set to be 7 days (static value) after the date when the record is created (dynamic value).

You may use pre-population rules in combination with a template to automatically populate text and memo text fields with static values, dynamic values, or string concatenation.

Usually, fields must be populated by a value of the same type. For example, a number field must be populated by a number. You may prepopulate text and memo-text fields with the following types of values:

- **Boolean—**Populate the field with **Yes** or **No** according to the boolean value (Boolean values are generally shown in the end user interface as check-boxes)
- **Date—**Populate the field with the system date and time formats
- **Number—**Populate the field with a number as text
- **Record**—Populate the field with the display string of the record (its name or number)

You may use pre-population rules in combination with a template to automatically populate number fields using the following calculations:

- Addition
- Subtraction
- Multiplication
- Division

- Percentage of a number
- Ratio between two numbers
- Total values from multiple child records, such as the sum of the **Amount** field from all line items of an invoice
- Number of items in a list, such as the total number of assignees for a matter record

1.1.10.3.2.2 Defining Fields with Literal Values

To define a field with a static value in a template

1. In the Designer window, from the **Go to** drop-down list select **Object Definitions**.
2. Select the appropriate object definition.
3. Select the **Templates** tab, and then select a template.
4. Open the **Records** tab.
5. Select the **General** folder.
6. Enter the **Order** of the field.

   The order is important only when the value of one field depends on the value from another field you are defining.
7. From the **Field** drop-down list, select the field you want to define.
8. From the **Value** drop-down list, select **Literal**.
9. Enter or select the appropriate value in the **Literal** field.

   For specific instructions and examples of how to enter a static value for different field types, see the following table.
10. Click **add more**.

Static Value Examples for X Fields

Attribute type	Instructions
Date	Enter the appropriate date manually or select it from the **Pop-up Calendar**. ***Note:*** *The format depends on the selected preferences.*
Text	Enter the appropriate text string.
Number	Enter the appropriate integer or real number.
Custom Object	Search for the appropriate record and select it using the search module.

Lookup Tables	Select the necessary item from the drop-down list.
Users	Select the necessary user from the drop-down list. ***Note:*** *Depending on the selected preferences, this field may be displayed as a user search module.*
Check-Boxes	Select or clear the check-box.
IID	Select the item from the drop-down list.

1.1.10.3.2.3 Defining Fields with Attribute Values

Attribute values are based on a value entered or selected in another field. They are also known as "dynamic" values. You may define dynamic values using Object Navigator.

To define a field with a dynamic value in a template

1. In the Designer window, from the **Go to** drop-down list select **Object Definitions**.
2. Select the appropriate object definition.
3. Select the **Templates** tab, and then select a template.
4. Open the **Records** tab.
5. Select the **General** folder.
6. Enter the **Order** of the field.

   The order is important only when the value of one field depends on the value from another field you are defining.
7. From the **Field** drop-down list, select the field you want to define.
8. From the **Value** drop-down list, select **Attribute**.

   Another drop-down list appears with applicable values, such as **Current Object**.
9. Click the **Object Navigator** icon to create the appropriate attribute path.

   For details, see Using Object Navigator.
10. Click **add more**.

1.1.10.3.2.4 Defining Fields with Formula Values

Formula values are derived from a combination of static and dynamic values. They are also known as "static-dynamic" values. You may define formula values using Object Navigator, Literal values, and operators such as & or +.

To define a field with static-dynamic values in a template

1. In the Designer window, from the **Go to** drop-down list select **Object Definitions**.
2. Select the appropriate object definition.
3. Select the **Templates** tab, and then select a template.
4. Open the **Records** tab.
5. Select the **General** folder.
6. Enter the **Order** of the attribute.

   The order is important only when the value of one field depends on the value from another field you are defining.
7. From the **Field** drop-down list, select the field you want to define.
8. From the **Value** drop-down list, select **Formula**.

   Formulas are only available for fields where a formula makes sense. For example, the activityItem field does not support a formula.

   A set of drop-down lists appears including Literal, Attribute, and available operators to create the appropriate static-dynamic value.
9. Select the appropriate values from the drop-down lists to define the appropriate field.

   Additional fields appear to allow you to select or enter the appropriate Literal or Attribute values.
10. Click the **Object Navigator** icon to create the appropriate attribute path.

    For details, see Using Object Navigator.
11. Click **add more**.

1.1.10.3.3 Adding Sub-objects to Templates

To automatically populate sub-objects using of wizards or rules, you may add the corresponding sub-objects to your template, such as categories, assignees, contact addresses, and so on.

Each sub-object consists of a set of fields that allow specific values to be assigned to them to create a valid record.

To add a sub-object

1. In the Designer window, from the **Go to** drop-down list select **Object Definitions**.
2. Select the appropriate object definition.
3. Select the **Templates** tab, and then select the appropriate template.

   The **General** tab of the template screen appears.

4. Select the **Records** tab.

   If no sub-objects have been defined in the selected template, only their main folder is available. That is, no subfolders have yet been defined.

5. Select the appropriate sub-object folder in the left pane.

   If the sub-object supports a default value, such as a default **Fax Number** in a Contact template, you may define it in the **Set Default** section using Literal and Attribute values.

6. Depending on the value type you want to assign to the default value, perform one of the following instructions:

   - Defining Fields with Literal Values
   - Defining Fields with Attribute Values
   - Defining Fields with Formula Values

7. If you want to add additional, non-default values, enter a text value in the **Friendly Name** field and click **add more** to create a subfolder.

   The value of the **Friendly Name** field is only used within the template. It does not appear in the resulting record.

8. If you have defined a **Friendly Name**, select its new subfolder.

9. Depending on the value type you want to assign to each field, perform one of the following instructions listed in step 6.

10. Click **Save**.

1.1.10.3.3.1 Adding Categories to Templates

To specify a category in a template

1. On the **Records** tab, click the **Categories** folder.
2. Select the appropriate default category from the **Category** list.
3. Click **add more**.

   The category has been added to the template and are added to records created by the wizard or rule that uses the template.

   You may now add custom fields of that category to the template.

To prepopulate the default category of object records, specify the default category with a static value in the template.

To specify the default category in a template

1. On the **Records** tab, click **Categories**.
2. Select the appropriate default category from the **Category** list.

3. Select the check-box next to the appropriate category in the list.
4. Click **set as default**.

   The default category has been specified in the template and is added as the default category in records created by the wizard or rule that uses the template.

1.1.10.3.3.2 Adding Assignees and Attendees to Templates

You may use pre-population rules in combination with a template to automatically populate the following elements:

- Task assignees
- Project assignees

To populate these elements, you must specify a user, group, contact, or contact group (Address Book). When specifying assignees or attendees, it is best to select a user instead of contacts or contact groups, because only contacts that are active users are assigned or included as an attendee. Adding contacts that are not active users does not cause an error to be displayed, but no assignee or attendee is added.

When users manually enter assignees in a task or project record, the system sets some assignee values automatically, such as **Status** and **Assigned On**, so you do not need to specify them in your template.

Assignees Batch Screen

You must define any required values in your template, which users "manually" enter, if you want to create a valid sub-object definition. For example, you must include the user's name and role.

Important: *In order for a sub-object to be valid you must define its required values.*

You may define assignees in a template and then set one as the default assignee.

To define an assignee in a template

1. Open the **Records** tab of the appropriate template.
2. Select the **Assignees** folder.

   Although Assignee is a sub-object of the Task object, task templates do not display a separate Assignee folder. You may define the Assignee field in the main General folder of task templates.
3. Enter a text value in the **Friendly Name** field and click **add more** to create a subfolder.

   The value of the **Friendly Name** field is only used within the template to name the subfolders. It does not appear in the resulting record.
4. Select the new subfolder.
5. Enter a value in the **Order** field.

   The order is important only when the value of one field depends on the value from another field you are defining.
6. From the **Field** drop-down list, select appropriate field name, such as **user**.
7. From the **Value** drop-down list, select **Literal**.

   Another drop-down list appears with applicable values.
8. Select **User Account**.

   Another drop-down list appears with available usernames. If there are more than 100 users, a search module appears.
9. Select the appropriate username.
10. Click **add more**.
11. To assign a role to the assignee, select **type** from the **Field** drop-down list.
12. From the **Value** field, select **Literal**.

    Another drop-down list appears with applicable values.
13. Select the appropriate role from the list.
14. Click **add more**.

Once you have defined at least one Assignee value in a template, you may select one as the default assignee.

To define the default assignee in a template

1. Open the **Records** tab of the appropriate template.
2. Select the **Assignees** folder.
3. In the **Set Default Assignee** section, select **Literal**.
4. From the drop-down list to the right of the **Literal** value, select the friendly name you specified for the folder with the appropriate assignee.

5. Click **add more**.

1.1.10.3.3.3 Deleting Sub-objects from Templates

Do not delete sub-objects that were added for the following reasons:

- To make custom fields created for a category (other than Root) accessible in a wizard.
- To prepopulate certain types of fields in the template that require a sub-object, for example, Main Assignee or Default Category.

To delete a sub-object from a template

1. In the Designer window, from the **Go to** drop-down list select **Object Definitions**.
2. Select the appropriate object definition.
3. Select the **Templates** tab, and then select a template.
4. Select the **Records** tab.
5. Select the sub-object's folder in the left pane.
6. Select the check-box next to the sub-object's **Friendly Name** field.
7. Click **delete**.

   The sub-object definition is removed from the template.

1.1.10.3.4 Adding Related Objects to Templates

Related objects are those objects whose records may be associated or linked to another object record and may be accessed directly from that object record. For details on the TeamConnect objects and their default relationships with one another, see About Objects.

Related objects in templates are associated with the main object in the following hierarchical ways:

- Child-parent
- Involved object
- Embedded object

Non-hierarchical related objects are not available on the **Records** tab of templates.

The **Records** tab of template screens displays a folder for each related object in the object definition. For example, a Dispute (custom object) template includes folders for the following related objects:

- Involved
- Task
- Appointments
- Expenses
- Accounts

- Embedded projects
- Child projects
- History

You may use templates to specify which related object records to create and which of their fields to populate with values.

Note: *Although Document is a related object to projects, you may not define documents on the* ***Records*** *tab of the template screen. For more information, see* ***Default Document Folder*** *in the General Tab on Template Screens table.*

Points To Remember

Consider the following points when defining related objects in a template:

- You have to add a separate related object for each associated record you want to be automatically created every time a new record is created. For example, if you need one account, one history record, five involved records, and five task records to be created, you must add twelve related objects in your template.
- You have the option of either creating a separate template for each related object or defining it within the main template.
  - If you need to define only a few default values for certain fields in a related object, you may want to do this within the main template, such as the **Due Date** and **Description** for a task.
  - If you need to define a more complex related object, with sub-objects, you have to create a separate template for it and the link it to the main template, for example, an account with its default category, posting criteria and a set of child accounts.
- You may also define a default document folder for each new record that is created based on the template. See **Default Document Folder** in the General Tab on Template Screens table.

You may define sub-objects on the **Records** tab of the template screen in the appropriate object definition.

To add a related object to a template

1. Make a list of all associated records you want to be created and linked to each record created based on the object template you are creating.
2. Decide which related objects you need to define for each associated record.
3. Decide what default values you want to be assigned to each attribute.
4. Decide for which related objects you might need to create separate templates.

   See Points To Remember.
5. Create the necessary separate templates to be linked to the selected related objects in the template you are creating.
6. Open the **Records** tab of the appropriate object template. See Using Template Features.

7. Select the appropriate related object folder in the left pane.
8. Enter a text value in the **Friendly Name** field and click **add more** to create a subfolder.

   The value of the **Friendly Name** field is only used within the template to name the subfolders. It does not appear in the resulting record.
9. Select the new subfolder.
10. Do one of the following actions:
    - Select **Use an existing template** if you want to use the fields and sub-objects that are already defined for the selected object within an existing template.
    - Select the **General** subfolder of the friendly named folder if you want to define fields and sub-objects that are already defined for the selected object within an existing template.

      The process of defining values for this option is the same as defining individual fields on the **Fields** tab of a template. The only difference is that you may not define fields that require sub-objects (for example, assignees). In this case, you must create a separate template for the related object and select the **Use an existing template** option to link it to the current template.
11. If you select the **General** subfolder option, perform to the following instructions, depending on the value type you want to assign to each attribute:
    - Defining Fields with Literal Values
    - Defining Fields with Attribute Values
    - Defining Fields with Formula Values

Adding Involved Objects

You may define Involved object values in a template to automatically add an involved party record and populate its **Contact** field. To do this, you must specify one of the following values:

- An active group or user as a dynamic value
- A group as a static value (the group must include at least one active user)
- A user's contact as a static value
- A contact group (Address Book) as a static value (the contact group must include at least one active user's contact record)
- A dynamic value ending with the following object attributes:
  - assigneeList(role).user
  - assigneeList(Any).user

Note: *Only active assignees may be used.*

1.1.11 Creating Wizards

A wizard is a user-interface utility that helps users create valid records by guiding them through the data-entry process, executing rules, populating defined values, and saving the record on completion.

When creating a new record, users have one or both of the following options:

- Creating the record manually by using the record screens to enter data in various fields. The record screens may not be organized in the same order in which users collect information when they create records.
- Using a wizard designed to guide users through the process of entering required information in predefined pages.

You may define rules for wizards to make sure users enter valid data. You may also create wizards that use templates, which automatically insert specific values into every record created from the wizard.

For more information about templates, see Using Templates.

Accessing Wizards

Once you define wizards, users can access them in the following locations in TeamConnect:

- The **New** button in the list view of a record—When you click this button, a drop-down list one or more wizards that are defined for the selected record type.
- The **New** button in a related object block—When you click this button, a drop-down list displays all of the wizards that are available for the selected related object.

1.1.11.1 Designing Wizards

When designing a wizard, you must consider various scenarios that users may face when creating records. For example, you need to determine the minimum information that must be captured by the wizard to quickly create a valid record, which fields may be automatically prepopulated, and how object definition rules might conflict with the wizard.

You must also design the layout of individual pages, page components, and page order. You may be able to reuse existing blocks of fields used in custom object record screens or create new blocks for the wizard.

Consider the following when designing wizards:

- General Considerations
- Writing a Wizard Specification Document
- Whether to Use Templates
- Designing Wizard Layout
- Identifying Wizard Flow and Page Transitions

Prerequisites

To create a set of useful and effective templates and wizards, you must meet the following requirements:

- Define your organization's business requirements and accepted practices.
- Understand TeamConnect's object model, whose attributes are described in Object Model: Read This First and the additional related reference tables.
- Understand template and wizard basics and the major TeamConnect components, such as sub-objects and related objects.

  For details, see:

  - Planning Templates
  - Adding Pages to Wizards
  - Introduction to Customization
- Write a wizard design specification based on your system design. It must specify custom fields, blocks, and so on, which need to be included in the wizard. For an example of this document, see the Wizard Design Specification Document Example table.

General Considerations

Use the following checklist to develop a general outline of the wizards you may need to create and what kind of information has to be collected:

- Determine the possible scenarios that may take place when a user must create a record.

  For example, in a legal department, a representative receives a call reporting a transaction. The representative attempts to create a Transaction record in TeamConnect. The appropriate wizard checks whether the user is a member of the Transaction Attorneys or Paralegal groups and preselects the wizard pages according to the type of information the current user is allowed to enter.
- Identify the information that must be collected when creating a record through each wizard and whether child records must be created and prepopulated with dynamic or static values.

  For example, certain wizards may assign a default category for the record, then create related records such as task, account, or audit-history records.
- Decide how many wizards are necessary to account for different scenarios, for example, one complex wizard with multiple pages and page transition rules or several smaller wizards, with each taking care of a specific scenario.

  For example, you may want to create separate wizards to collect transaction details depending on the specified categories, and separate wizards for different involved party roles or child records.

Once you have a general outline for the wizard, you might want to write a specification document detailing all the wizard requirements. If you decide to create more than one wizard, the best practice is to create a separate document for each wizard. See Writing a Wizard Specification Document for more details.

1.1.11.1.1 Writing a Wizard Specification Document

A wizard specification document details all wizard requirements by determining the following information for each step (page) of the wizard:

- Object for which the wizard and any associated templates must be created.
- System and custom fields, or blocks of fields, which capture the appropriate information that needs to be gathered.
- Required fields as specified in the object definition, such as:
  - The parent project field.
  - Fields whose values are used in the unique ID or name patterns.
  - Fields that are required by the rules created for the selected object on the Create trigger--create rules are not triggered until the user reaches the end of the wizard, which is when the record is created.
  - Required custom fields to be included in the object view, and so on.

  For more details on required fields, see Required Fields.
- Additional fields necessary only in the wizard. For example, parameter fields might require the user to enter a **Yes**, **No**, or **Unknown** value to determine the next wizard page. Such fields are not saved or displayed in the resulting record. For details, see Defining Parameters.
- Default values for certain fields.
- Additional rules to ensure the needed values are entered through the wizard.

  For example, if a user selects **Yes** in response to a wizard question, the user must make at least one selection from a set of later-displayed check-boxes. This requirement might be accomplished using a custom block or by creating a page transition rule that directs the user to an end page.
- Error messages displayed for the user if a page transition rule is triggered.
- Prompts to help the user enter information for the fields in the record.
- The field order on each page.
- Page transitions, including dependencies on the input from the user.

The following table provides an example of a wizard requirements specification document for an insurance claim.

Wizard Design Specification Document Example

#	Wizard question	Field type	Field label	Display in record	Wizard page component	Comments
1	What is the caller's first name and last name?	Text	None	No	Parameter	If the caller is a third party, an action must be

						written to add the names to the involved party record.
2	What is the caller's phone number?	Text	None	No	Parameter	If the caller is a third party, an action must be written to add the number to the new contact record.
3	Is the caller insured or third party?	Radio Button	None	No	Parameter	Insured or third-party values • If Insured, go to #4 • If third party, go to #16
4	What is the insured's policy number?	Search Module	Policy Number	Yes	System Field	This is the name or number of the policy.
5	When did the accident occur?	Date & Time	Date of Loss	Yes	Custom Field	This is the date of the accident.
6	Where did the accident occur? 1 Location 2 State	3 Text field 4 List	5 Accident Location 6 Location State	7 Yes 8 Yes	9 Custom Field 10 Custom Field	This is the physical location (for example, intersection of 5th and Main St.) Use State List lookup table
7	Did the accident involve an intersection?	Radio Button	None	No	Parameter	Yes or No Value • If Yes, go to #8 • If No, skip to #9
8	What Type of Intersection was involved?	List	Intersection Type	Yes	Custom Field	Default to No.

9	Did the accident involve traffic controls or signage?	Radio Button	None	No	Parameter	Yes or No Value • If Yes, go to #10 • If No, skip to #11
10	What Type of Traffic Control was involved?	List	Traffic Control	Yes	Custom Field	Default to No.
11	Please describe the accident.	Memo Text	Description of Loss	Yes	Custom Field	For example, the insured vehicle failed to yield right of way and made a left turn in front of claimant vehicle.
12	Are there any Area Risk Factors that apply to this accident?	Radio Button	None	No	Parameter	Yes or No Value • If Yes, go to #13 • If No, skip to #14
13	Indicate the Area Risk Factors that apply to this accident.	Check-Boxes	Multiple - Area Risk block	Yes	Block	A rule must be added to require at least one selection.
14	Additional required information					

1.1.11.1.2 Whether to Use Templates

Templates define what values are automatically populated for each type of record and what related records are automatically created. You may create wizards in association with a template or independently of a template.

Note: *The information listed in your wizard specification document helps you decide whether using a template is worth the additional effort (see Writing a Wizard Specification Document).*

If any of the following conditions exist, you must create a template for your wizard:

- Sub-objects that need to be added to the record. For example, you need to add sub-objects if you are:
  - Using custom fields of a category other than root in your wizard

- Setting a certain non-root category, such as **Lawsuit**, as a default category for all your records created from the wizard
- Adding several assignees to the record and setting one of them as the main one by default

- Fields that may be automatically prepopulated in every new record, such as the **Default Category** or the **Main Assignee**.
- Related records that need to be automatically created together with every new record, such as tasks or accounts.

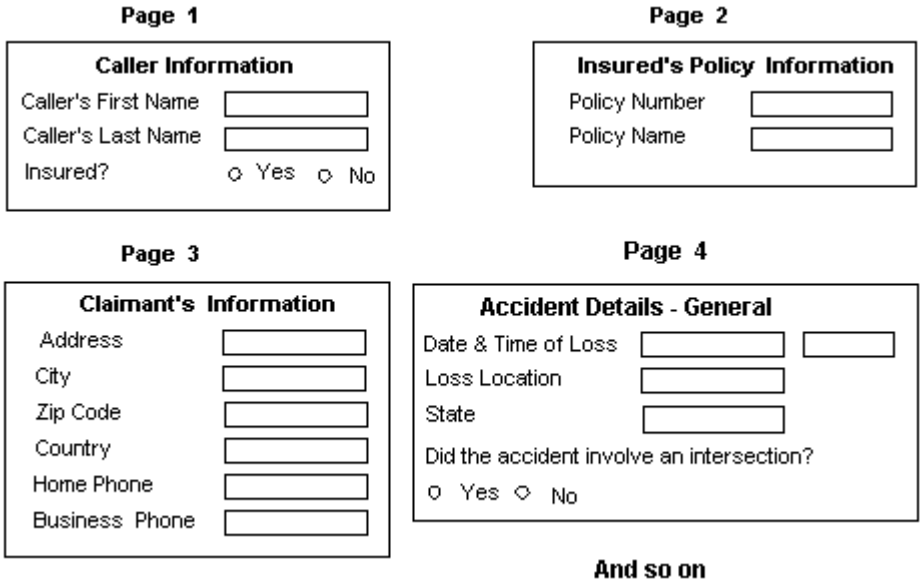
  You may create separate templates for associated records, such as accounts or invoices.

If you only need to prepopulate individual fields that do not require sub-objects, such as descriptions, dates, number fields, and so on, they may be prepopulated with the help of simple actions in the wizard itself without the need for a template.

1.1.11.1.3 Designing Wizard Layout

Once you specify the information that you want to capture, you may design the layout of your wizard.

Organize the fields and any questions into separate groups as you would expect them to appear in each page of the wizard, such as the following example:



Wizard Page Layout Example

Tip: *If necessary, you may add wizard prompts to wizard pages as a component of the type Text.*

Through the process of laying out page components, you may gradually determine the contents of each page, such as individual custom and system fields, blocks, text, or parameters. For more details, see Defining Page Components in Wizards.

Through this process, you may decide to create new custom blocks for certain wizard pages.

Tip: *You may reuse the blocks created for object views of the object for which you are creating the wizard, or create new custom blocks for wizard pages, for example, to include parameters in them.*

1.1.11.1.4 Identifying Wizard Flow and Page Transitions

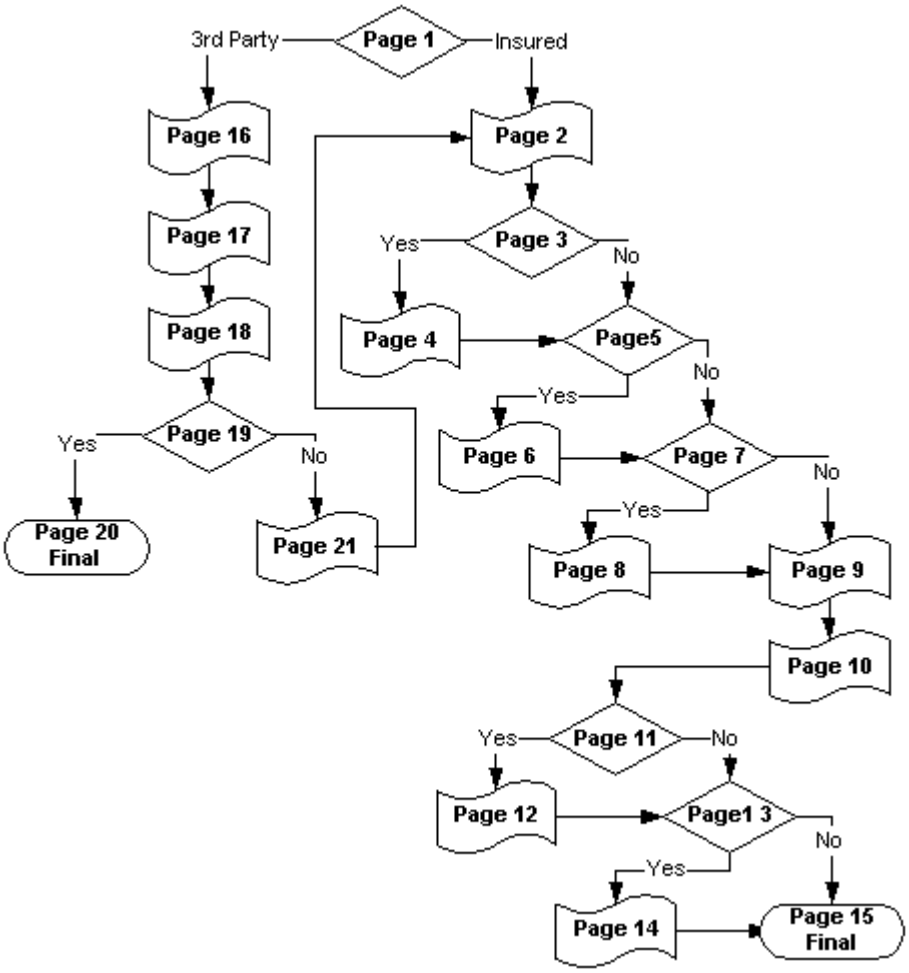
After you have decided on the page layout, you need to make sure the pages appear in the appropriate order to the user creating a record. Plan your page order as follows:

- Finalize the general flow of the wizard. Namely, define the sequence in which the wizard pages follow each other through page transitions.

  For example, in the following diagram, an alternative sequence may result from Page 1 if the involved is a third party.

- Identify the pages that require page transition rules and the qualifiers. These pages are marked by the diamond shapes in the following diagram.

- Determine how many final pages may be required. These requirements include pages that allow the user to exit the wizard and pages that allow them to save the record, and dependencies such as missing information, limited rights, and so on. These pages are marked by the oval shapes in the following diagram.



Wizard Page Transition Diagram Example

- Make a list of the page transition rules you have to create for page transitions based on the input data. You must:
  - Identify the parameters used in qualifier items (conditions) for each rule.
  - Identify the pages to which you want the user to be directed in each of the possible alternative flows.

  The order you indicate for the wizard pages, as you add them to the wizard, is always set as default order.

1.1.11.2 Creating Wizards

Once you plan your wizard, determine its alternative flows, its wording, and the necessary fields and blocks, you are ready to create wizard pages. For each page, you must define individual components, define actions, and specify the necessary page transitions.

The following general procedure summarizes the process that is necessary for creating a wizard and refers to other sections in this guide for details:

1. Create a wizard specification document and complete all the prerequisites outlined in Designing Wizards.

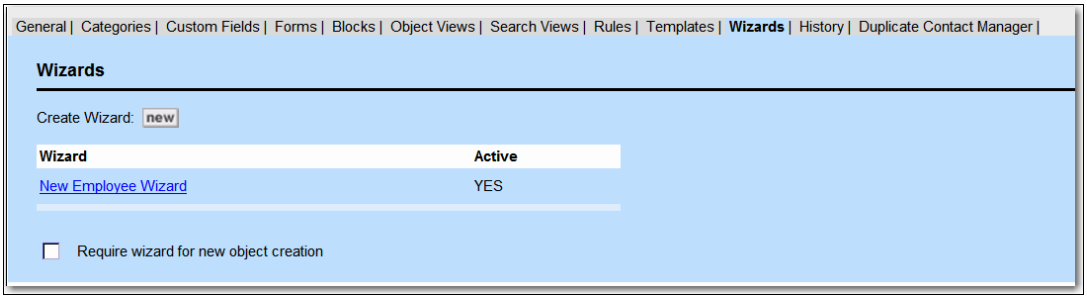
2. Determine whether you are going to use a template with your wizard. See Whether to Use Templates.
3. If necessary, create the appropriate templates. See Defining General Wizard Information.
4. In the Designer window, from the **Go to** drop-down list select **Object Definitions**.
5. Open the appropriate object definition.
6. Click the **Wizards** tab, and then click **New**.
7. Define the general wizard information as described in the General Tab on Wizard Screens table.
8. Click the **Pages** tab and add all appropriate pages as described in Adding Pages to Wizards.
9. Click the **Page Components** tab. Define and add all wizard page components to each page as described in Defining Page Components in Wizards.
10. Click the **Actions** tab and define any actions for the appropriate wizard pages as described in Using Actions in Wizards.
11. Click the **Page Transitions** tab and modify the default page transition order by adding page transition rules to the appropriate wizard pages as described in Defining Page Transition Logic.
12. Click the **General** tab and select the **This wizard is active** check-box and/or the **Require wizard for a new object definition** check-box if necessary.
13. Save the wizard and thoroughly test it. See Testing and Troubleshooting Wizards.

1.1.11.2.1 Opening the Wizard Screen

To open the Wizard screen from an object definition

1. Select **Object Definitions**, in the Designer window, from the **Go to** drop-down list.
2. Select the appropriate object definition.
3. Open the **Wizards** tab.

   The **Wizards** tab opens with a list of existing wizards defined for the selected object (if any).



Wizards Tab

4. Do one of the following actions:
   - To open an existing wizard, click the respective link.
   - To create a new wizard, click **new**.

The corresponding **Wizard** screen opens with its **General** tab displayed by default.

The Wizard screen consists of the following tabs:

- **General**—Displays the name of the wizard, the object it is defined for, and its template (if any). You can also activate the wizard and specify that users may only create objects of this type using a wizard. See Defining General Wizard Information.
- **Pages**—Allows you to view and create all the pages for the wizard. See Adding Pages to Wizards.
- **Page Components**—Allows you to view and define the contents of each page in the wizard. See Defining Page Components in Wizards.
- **Actions**—Allows you to define the necessary actions you may want for the selected wizard pages. See Using Actions in Wizards.
- **Page Transitions**—Allows you to check what page follows the currently selected page and to define the appropriate page transition rules. See Defining Page Transition Logic.

1.1.11.2.2 Defining General Wizard Information

The **General** tab of the **Wizard** screen displays:

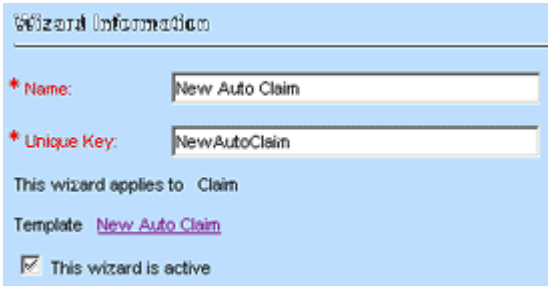
- The wizard's name
- The unique key of the wizard within the object
- The name of the object it is defined for
- The name of the template it is associated with (if any)
- Whether the wizard is active
- Whether to provide only a wizard to create objects of this type

To define general wizard information

1. Access the **General** tab of the wizard screen by following the instructions in Opening the Wizard Screen.

   The **General** tab of the wizard screen opens by default.

2. Define the information on the **General** tab according to the information in the General Tab on Wizard Screens table.



General Tab on Wizard Screens

The following table describes the items on the **General** tab of the **Wizard** screen.

Note: *The fields in the table below are the default fields that appear for all object definitions. You may see additional fields that only apply to the selected object definition.*

General Tab on Wizard Screens

Item	Description
Name	Enter a name that uniquely identifies the wizard and describes its purpose. This name appears to the users when they start to create a new record.
Unique Key	Enter an alphanumeric combination to uniquely identify the wizard within the object definition. It must be 50 or fewer characters in length and must not contain special characters such as spaces, underscores, or punctuation marks. After you save the wizard, the unique code appears as read-only text and you cannot change it. Wizards within the same object definition cannot have the same unique key. However, wizards belonging to different object definitions may have the same key.
This wizard applies to	Automatically displays the name of the object that the wizard is created for.
Template	Select one of the available templates you want to use in the wizard (if applicable). ***Note:*** *If you intend to use a template with the wizard you are creating, make sure you create and define the template first. See* *General Template Information*.
This wizard is active	Select this check-box to activate the wizard. Clear the check-box to deactivate the wizard. This field is available only after at least one page is added to the wizard. Until then, instead of this field, the following message appears: "The wizard is currently inactive. To be able to activate it, please add a page." ***Caution:*** *Even when the check-box appears, do not activate the wizard until it is completely finished and ready for testing.*
Require wizard for new object	Select this check-box if you want users to only have the option to create a record using a wizard. When users click the **New** button, the

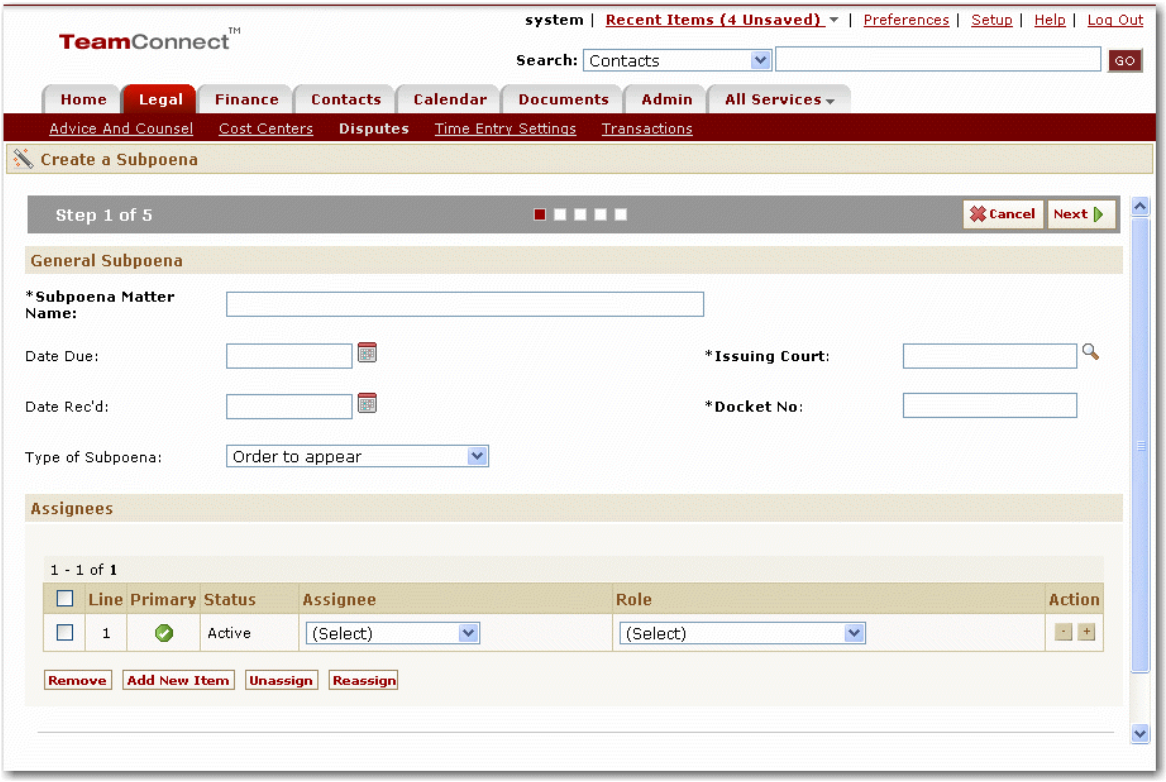
definition	wizard opens automatically. Clear the check-box if you want users to have the options to create a record using a wizard and create a record manually using a blank record. This field is only visible if at least one active wizard is defined for this object.

1.1.11.2.3 Adding Pages to Wizards

Wizard pages are represented by separate web page screens in the end-user interface. To create wizards, you add pages, define their individual contents, and specify page transitions. Each page may have a number of fields and may include a **Next** button, a **Back** button, a **Cancel** button, and a **Finish** button.

All of the necessary wizard pages must be added on the **Pages** tab of the corresponding wizard screen in Designer. You must define the order in which the pages must follow each other by default.

Depending on the number of alternative flows in the wizard and their page transitions, you may need to define more than one final page, as described in Identifying Wizard Flow and Page Transitions.

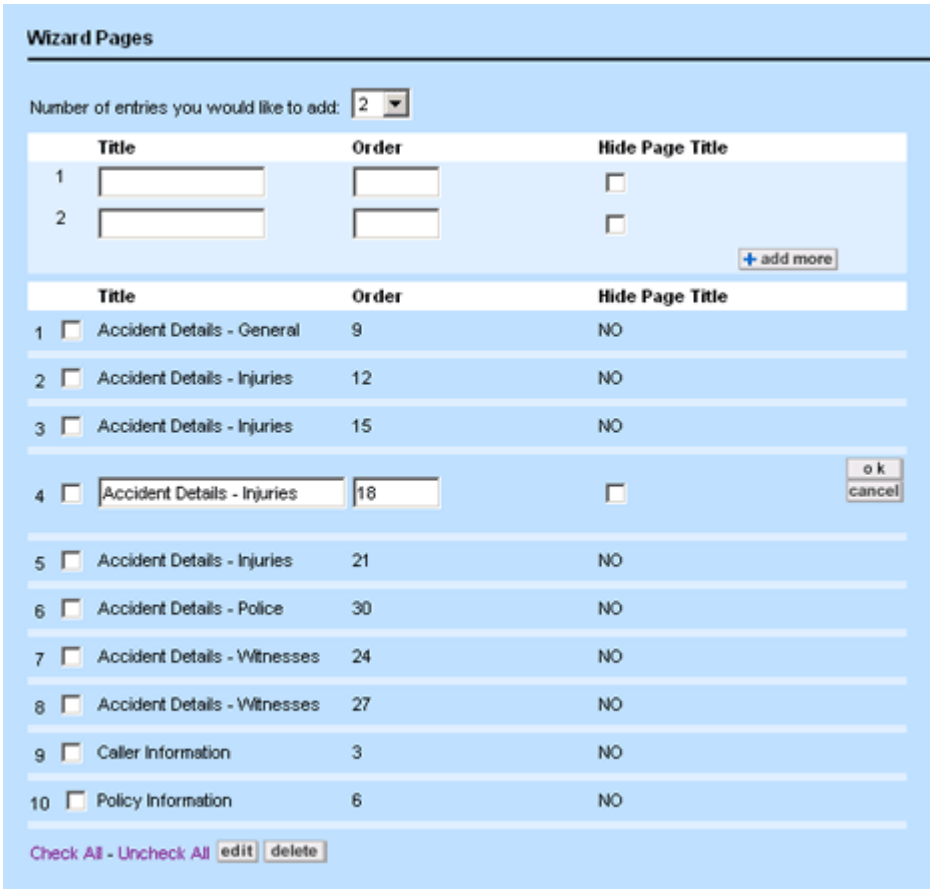


Example: Wizard Page

To add a page to a wizard

1. Open the appropriate wizard screen and select the **Pages** tab.

The **Pages** tab opens.



Wizard Pages Tab

2. Select the **Number of entries you would like to add** from the drop-down list.
3. For each data entry row, define the page title, its order, and whether the page title must be hidden in the wizard as described in the Wizard Pages Tab table.
4. Click **add more** and repeat step 3 until you've added all of the wizard pages.

For each page that you define, you must add the fields and blocks that you want to appear on each page. Click the **Page Components** tab to define the appropriate components for each page of the wizard. For instructions, see Defining Page Components in Wizards.

The following table describes the items on the **Pages** tab of the wizard screen.

Wizard Pages Tab

Item	Description
Title	Enter a name for the page, This name appears at the top of the wizard page as its heading. See the Wizard Page image for an example.

Order	Enter a number to specify the default order in which the page is to appear in the wizard. ***Tip:*** *When defining the page order, you may want to skip a few numbers before each page (for example, use 3, 6, 9, 13, 18, and 21,) in case you need to insert a few additional pages later if your wizard design changes.* This is also the order in which the names of the pages you are creating appear in **Current Page** drop-down list of the **Page Components**, **Actions**, and **Page Transitions** tabs and in the left pane when users are using the wizard. The default order that you create for the wizard pages may be modified through the appropriate page transition rules (Defining Page Transition Logic).
Hide Page Title	Select this check-box if you want the page title to be hidden. By default, this option is not selected. If you applied a block template to custom blocks, and your custom blocks have their block titles specified in the XML, this option prevents the title from being displayed twice.
add more	Click to add the defined page to the page list.
Page List	Displays all pages added to the selected wizard page. Select the check-boxes next to the components that you want to delete or edit and click the appropriate command button at the bottom.

1.1.11.2.4 Defining Page Components in Wizards

You may define the contents of each wizard page on the **Page Components** tab of the wizard screen, which provides the following types of components:

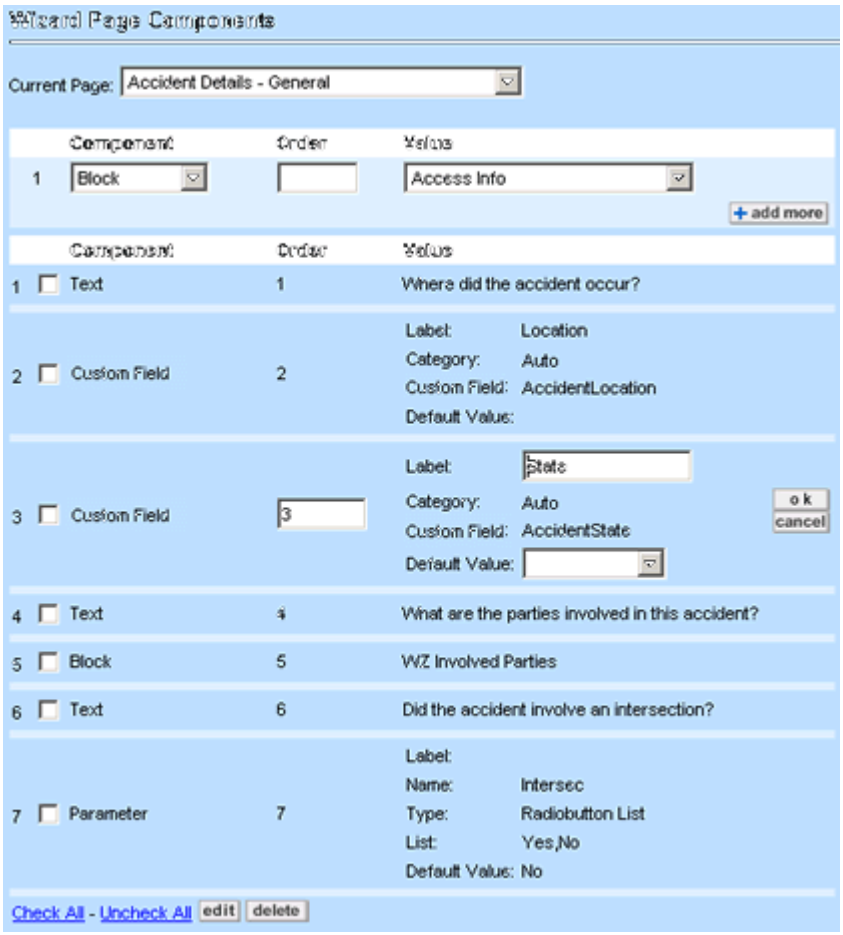
- Text
- Block
- System Field
- Custom Field
- Parameter

To use the **Page Components** tab, you must first create wizard pages.

Note: *If duplicate contact checking is enabled and you have created a wizard that creates contacts using the Contact search module (Create New Contacts pop-up window), under certain conditions certain contacts may not be created. See the Contacts for more information.*

To define a page component in a wizard

1. Open the appropriate wizard and select the **Page Components** tab.



Wizard Page Components Tab

2. Select the wizard page you want to add components to from the **Current Page** drop-down list.
3. From the **Component** list, select the component you want to add to the selected page.

   The available components and their values are described in the Wizard Page Component Value Fields table.
4. In the **Order** field, enter an integer to indicate the order in which the component must appear in the wizard page.
5. In the **Value** area, enter or select the appropriate values for the selected component.

   Different fields appear based on the **Component** that you selected.
6. Click **add more**.
7. Repeat steps 3 through 6 to add other components to the selected page and click **Save**.

Repeat steps 2 through 7 for each page in the wizard.

For an example of how the components shown in the Wizard Page Components Tab image appear in the resulting wizard page, see the Wizard Page Component Value Fields table.

The following image describes the fields in the **Page Components** tab.

Wizard Page Components Tab

Field or button	Description
Current Page	Select the wizard page you want to define components for. All pages in this field are listed in the order assigned to them on the **Pages** tab. If no pages have been created for the wizard, **(No Item Selected)** appears in read-only text.
Component	Select the component you want to add to the selected wizard page.
Order	Enter an integer to indicate the order in which the components must follow each other when displayed in the wizard page. ***Tip:*** *When defining the order, you may want to skip a few numbers before each component (for example, 3, 6, 9, 13, 18, 22, and so on.) in case you may need to insert a few additional components later, for example, if your page design changes.*
Value	Enter the appropriate values in the corresponding fields displayed in the **Value** column. The number and type of fields varies depending on the component selected and the type of field it is represented by on the wizard page. The **Default Value** is not a required field in most of the cases, because it depends on the user input. For details on various **Value** fields for different components, see the Wizard Page Component Value Fields table.
add more	Click to add the new item to the list of page components.
Page Component List	Displays all components added to the selected wizard page. Select the check-boxes next to the components you want to delete or edit then click the appropriate command button at the bottom.

The following table describes the component values available on the **Page Components** tab.

Wizard Page Component Value Fields

Component	Description

Text	Type the **Value** text that you want to appear in the wizard page, for example, text prompts for the user. (optional) For localization purposes, enter a **Unique Key** that you want to assign to this text field. ***Tip:*** *You may also want to use this component instead of field labels.*
Block	Select the system or custom block that you want to appear in the selected wizard page. The contents of this drop-down list vary depending on the object you are creating the wizard for. For more details on blocks in TeamConnect, see Blocks.
System Field	Complete the following steps to add a system field from the object for which you are creating the wizard: 1. Type a **Label** for the system field, as you want it to appear in the wizard page. The label does not have to be the same as in the system object views. 2. Select the appropriate attribute in the **Name** drop-down list. 3. Enter a **Default Value** for the select attribute, if any. The **Default Value** field varies depending on the attribute selected. Labels for fields are optional because they may be self-explanatory or explained by the wizard prompts and questions. You may add only the system fields of the main object whose records are created through the wizard—not those of related objects. If you associate a template with the wizard and add a sub-object to it, you may select its main assignee or default category.
Custom Field	Complete the following steps to add a custom field from the object for which you are creating the wizard: 1. Create a block for the desired custom fields before creating the wizard. 2. Type the appropriate **Label** for the custom field as you want it to appear in the wizard page (optional).

	The label does not have to be the same as in the custom object views. 3. Select the block created for the desired custom fields. 4. In the **Custom Field** drop-down select the name of the field you want to add to the selected wizard page. 5. (Optional) Enter the appropriate **Default Value** for the selected custom field. The **Default Value** field varies depending on the custom field selected. You may add the custom fields created for the object's root category only. However, if you associate a template with the wizard and add categories other than Root, then their respective custom fields are also available for you to add to the wizard. For details on adding a category to a template, see Adding Sub-objects to Templates.
Parameter	Parameters allow you to add items that you may need to use as qualifiers for page transitions or to perform certain actions on the object attributes. Parameter data is not stored in the database and is only used within the wizard. Complete the following steps to add a parameter to the wizard page: 1. Type the appropriate **Label** for the Parameter as you want it to appear in the wizard page. 2. Select the **Type** of the Parameter you want to add to the page. For details on parameters, see Defining Parameters. 3. Type a unique name for the parameter you are creating. This is the name that are required when you use the parameter in page transition rules or page actions. 4. Check **Display in blocks only** if you intend to use parameters in a custom block. Otherwise, the parameter appears twice on the page, in the block and as an individual field. 5. Select the Custom Lookup Table that will contain the list for this parameter. This applies to check-boxes, radio buttons, and drop-down lists that are used in custom blocks. 6. (Optional) Enter the appropriate **Default Value** for the selected Parameter. The **Default Value** field varies depending on the type of parameter selected.

System Field **Custom Field** **Parameter**	of type **String** (**Text Fields**)	For system fields, custom fields, and parameters represented by a Text Field in the wizard page, there are two additional fields in the **Value** column: • **Field Size** • **Maximum Length** These two values allow you to specify how many characters the text field may display (**Field Size**) and how many characters it may contain (**Maximum Length**).
	of type **Long String** (**Memo Text Fields**)	For system fields, custom fields, and parameters that are represented by a Memo Text Field in the wizard page there are two additional fields in the **Value** column: • **Columns** • **Rows** These two values allow you to specify the size of the memo text box vertically (**Rows**) and horizontally (**Columns**).

1.1.11.2.4.1 Defining Wizard Parameters

Parameters allow you to add items to be used as qualifiers for page transitions or to perform operations on the object attributes. Parameter data is not stored in the database and is only used within the wizard. Parameters may be simple, such as Number, String, or List, or use functionality that is stored in the database.

For each parameter that you add on the **Page Components** tab of the wizard definition, you may enter:

- A **Label** (required)
- The **Type** (required)
- A unique **Name** (required)
- **Custom Lookup Table** (required for check-boxes, radio buttons, and drop-down lists)
- The **Default Value** (optional)

The unique name is used in the following locations in the wizard screen:

- On the **Qualifier** tab of the page transition rules in the drop-down list for the **Left Argument**
- In the **Right Argument** if you select **Attribute**
- On the **Actions** tab in the **Parameter** drop-down list
- In the name attribute of the `<tc:wizardParameter name>` and `<tc:wizardParameterLabel name>` tags, when the parameter is included in a custom block. For details, see Block Tags.

Caution: *When you are including a parameter in a custom block that also contains custom fields, the parameter names and field names must be unique. Otherwise, errors may occur in page transition rules.*

If you plan to include the parameter in a custom block, make sure to select the **Display in blocks only** check-box, or it appears twice on the wizard page.

Database Parameters

Complex parameter values are stored in TeamConnect's database. The following parameter types used in wizard page components are mostly system lookup tables that appear in drop-down lists in the wizard:

- Activity Item
- Address Type
- Area Item
- Category
- Check-Box List
- Contact Relation Type
- Country Item
- Drop Down List
- Email Type
- Fax Type
- Group
- Internet Address Type
- Invoice Rejection Reason
- Integration Message Queue
- Phone Type
- Project Relation Type
- Radio Button List
- Resource Type
- Skill Type
- String
- Territory Type
- User

Depending on system and user preferences, these items may also appear in an applet. The **User** parameter type may appear in a search module.

If you select any of these parameter types, the **Default Value** field in the **Value** column on the **Page Components** tab appears a drop-down list with the corresponding lookup table items so you may select a default value. In the resulting wizard page, the selected parameter is represented by the same drop-down list, applet, or search module with its respective items.

The following parameter types used in wizard page components are used to create a corresponding search module field, where the user may select the necessary record:

- Account
- Contact
- Custom Object

Transient Parameters

The following parameter types used in wizard page components have transient values. That is, they are not stored in the database and are created within the wizard itself:

- **Simple types**—Depending on the type of user input, select the appropriate parameter to create the corresponding type of field in the wizard page.
  - Boolean
  - Date
  - Date&Time
  - Decimal
  - Integer
  - Memo Text
  - (Text) String
- **List types**—Add list parameters to present the user with simple options on the wizard page, such as **Yes**, **No**, or **Unknown**.
  - Check-Box List
  - Drop Down List
  - Radio Button List

To define simple list type parameters

On the **Page Components** tab of the wizard definition, select an option in the **Custom Lookup Table** drop-down list.

Tip: *Automated actions may populate a list parameter with values retrieved from records in the database. For example, you might display a list of outside counsel with certain rates and skills next to corresponding radio buttons.*

Points To Remember

The following list summarizes wizard parameters:

- All parameters have transient values that exist only within the wizard in which they are defined.
- Parameter data is not stored in the TeamConnect database.
- Use its value to perform certain actions or as a qualifier for page transitions.
- Parameter values may be used in the wizard page for which you defined it and a any of the subsequent pages in the wizard for actions or page transitions.

  For example, if you add a parameter of type Integer to page 3 to obtain the number of days required to complete a task, on page 6, you may use its value to create an action to set the **Due On** date for the task.
- Parameters types are listed in the **Type** drop-down list in the **Value** column of the **Page Components** tab.
- Like custom fields, you may include wizard parameters in custom blocks to control their layout on the page.

  You may do this by using the `<tc:wizardParameter>` and `<tc:wizardParameterLabel>` tags. For details, Block Tags.

Important: *If you intend to use a parameter in a custom block, make sure the* ***Display in blocks only*** *check-box is selected. Otherwise, the parameter appears twice on the page, in the block and as an individual field.*

1.1.11.2.5 Using Actions in Wizards

Wizard actions are operations that allow you to create appropriate behavior in a wizard, such as creating additional records, inserting values, accessing other records, or performing calculations to assign values to object attributes. Depending on your wizard design, applying actions to wizard pages is optional.

All wizard actions may be classified into the following groups based on:

- **When they are executed**
  - **Initial Actions**—Executed when a wizard is started by the user but before the first page opens and may not be defined as a simple action.
  - **Page Actions**—Executed when the user clicks **next** on wizard pages for which they are defined and may be defined as a simple or automated action.
- **How they are defined**
  - **Simple actions**—Defined through the user interface and are limited to a predefined set of operators
  - **Automated actions**—Defined in Java class or JavaScript files and may perform any operations on record values without limitation

The following table summarizes the relationships between the four types of wizard actions:

Wizard Action	Simple	Automated

Initial		**X**
Page	**X**	**X**

Important: *You may only use one initial action file per wizard and only one automated page action file per page. However, you may define several simple actions for a page.*

To define an initial action for your wizard or to add an automated action to a page, you must first create each page, and upload the Java class or JavaScript file to the appropriate document folder for the object.

To define a simple page action, see Defining Simple Actions.

1.1.11.2.5.1 Applying Automated Actions

Automated actions may perform any operations on record values without limitation. In wizards, these actions include initial actions and page actions.

You must define automated actions in their respective Java class or JavaScript files and upload them to the appropriate directory in the Documents area. Otherwise, they are not available on the **Actions** tab of the wizard screen.

Although automated actions may perform simple operations, you may use them to create more complex logic that may not be defined through the user interface, such as:

- Validating or verifying information entered by the user
- Opening existing records at the end of a wizard
- Inserting values in parameters of the type List.

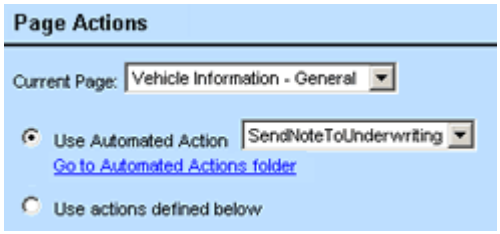
Note: *When you insert values in these parameters, you insert data gathered through the wizard into the appropriate screens of the object record. By default, duplicate contact checking is not performed when custom Java code creates or updates contacts. See the ContactService.readDuplicateContacts() method in the API Javadocs to use duplicate contact checking functionality.*

To apply an automated action to a page in a wizard

1. Upload the Java class or JavaScript files to the **Automated Actions** folder.
2. Open the **Actions** tab in the appropriate wizard.
3. In the **Page Actions** section, select the wizard page you want to apply the action to in the **Current Page** drop-down list.
4. Select the **Use Automated Action** radio button.

   The **Use Automated Action** drop-down list opens.
5. From the **Use Automated Action** drop-down list, select the file with the action you want to be applied to the selected wizard page.

6. Click **Save**.



Automated Action Option on Wizard Actions Tab

Points To Remember

- You may use only one automated action file per page.
- Automated actions are optional.
- The **Automated Actions** folder location in the Documents area is:

  Top Level/System/Object Definitions/objectName/Rules/Automated Actions/

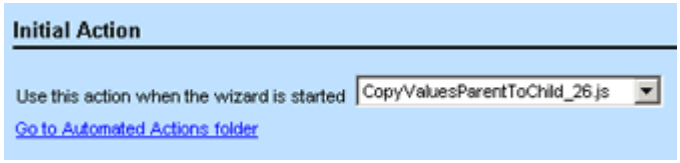
For information on how to create automated actions, see Writing Automated Actions.

1.1.11.2.5.2 Specifying Initial Actions

Initial actions are used to run an initialization routine when a wizard is started by the user but before the first page of a wizard opens. For example, these actions may compute the values inserted from the template associated with the wizard.

To specify the initial action for a wizard

1. Make sure the necessary files are uploaded to the **Automated Actions** folder of the object definition.
2. Open the **Actions** tab in the appropriate wizard.
3. In the **Initial Action** section, from the **Use this action when the wizard is started** drop-down list, select the file with the action that you want to be executed.
4. Click **Save**.



Initial Action Section on Wizard Actions Tab

Points To Remember

- Initial actions are optional and depends on your wizard design.
- Initial actions must be defined as automated actions.

- You may only define one initial action per wizard.

See also Execution and Population Order.

1.1.11.2.5.3 Defining Simple Actions

Simple actions are operations, such as calculations, string concatenation, and so on, that allow you to assign specific values to object attributes. These actions are always defined through the user interface and are limited to the number of the predefined operators available for each value type.

You may assign values to object attributes belonging to the object's related or child objects—provided you have defined the objects in a template and associated the template with the wizard. Once defined in the template, the sub-objects are listed in the **Object** drop-down list of the **Actions** tab.

You may assign the following types of values:

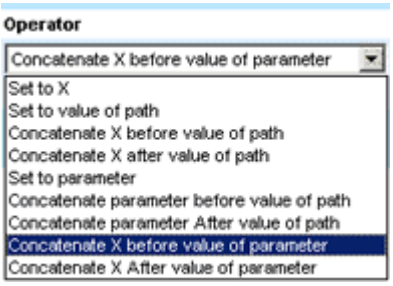
- Static values
- Values of wizard parameters
- Values of other attributes using a path in the object model

Use simple actions in a wizard when:

- You want to populate certain fields in the main record or its associated records with the values of parameters.
- When you want to overwrite the values assigned in a template with the ones specified on the **Actions** tab.
- You are not planning to use templates with your wizard, yet need to have certain fields of the main record prepopulated with default values.

Important: *Without a template you may not assign values to related objects or sub-objects.*

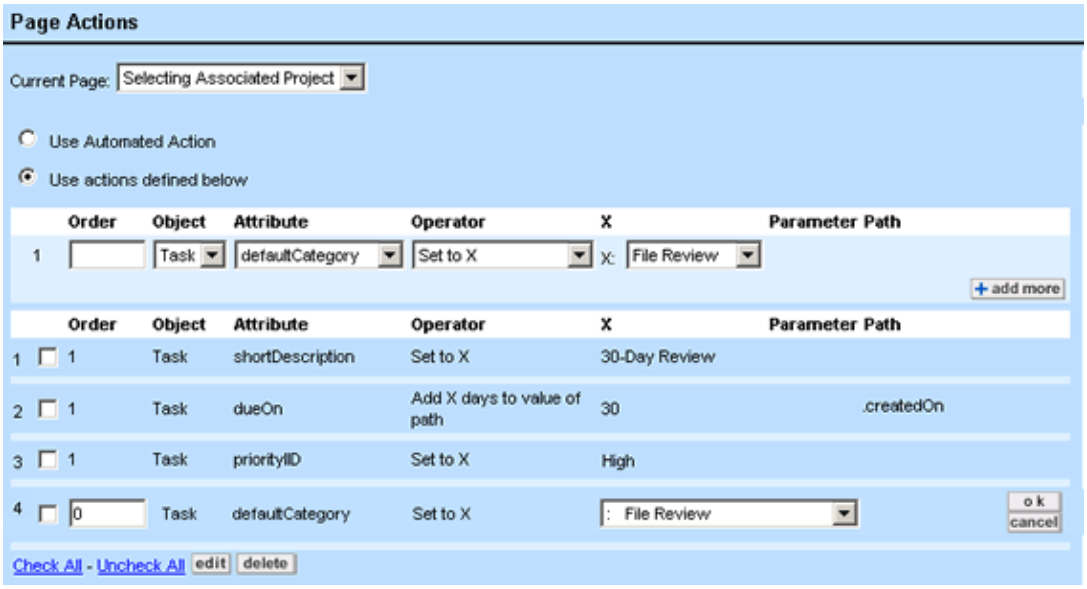
Simple actions are performed with the help of operators preselected for each value type. Depending on the attribute type, these operators may include the values shown in the following image and listed in the Operator Options for Different Attribute Types table.



Operator Options

To define a simple action for a page in a wizard

1. Open the **Actions** tab in the appropriate wizard screen.

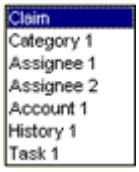


Simple Actions in the Page Actions Section of the Actions Tab

2. In the **Page Actions** section of the **Actions** tab, select the appropriate page from the **Current Page** drop-down list.
3. Select the **Use actions defined below** radio button.

   The screen refreshes to display data entry fields necessary for defining a simple action.
4. In the **Order** field, enter a number to indicate the order in which the action must be executed.
5. From the **Object** drop-down list, select the appropriate object for which this action is being defined.

   If you have associated a template with the wizard that defines child objects, these objects are listed in the **Object** drop-down list. They are numbered the same way as they are in their respective template screens, for example, Task 1, Task 2.



Object Drop-down List

6. From the **Attribute** and **Operator** drop-down lists, select the appropriate attribute and the operator.

   Depending on the attribute type, operators may include a set of operators listed in the Operator Options for Different Attribute Types table, in addition to another set of the same operators using the value of a parameter. For example, instead of **Path** or **X**, the operators could be **Set to Parameter**, **Set to Value of Parameter**, **Subtract X from Value of Parameter**.

7. Depending on the field value type of the selected attribute, enter the appropriate values for **X**, **Parameter**, or **Path**.

   To learn more about setting a path, see Paths.

8. Click **add more**.

9. Repeat steps 4 through 7 to add other actions to the selected page, if necessary, and click **Save**.

The following table describes the Action options on the **Actions** tab.

Simple Actions Options on Wizard Actions Tab

Field or button	Description
Current Page	Select the wizard page from which you want to define actions. ***Note:*** *All pages in this field are listed in the order assigned to them on the* ***Pages*** *tab.*
Use Automated Action	Select this option to apply an automated action to the selected wizard page. The **Page Actions** section refreshes to display a list of available automated actions created for the selected object (for example, the Automated Action Option on Wizard Actions Tab image). For more details, see Applying Automated Actions.
Use actions defined below	This is the default option. Select it to define your own simple actions. The **Page Actions** section refreshes to display data entry fields where you may assign appropriate values to the selected fields through their respective attributes. For details, see Defining Simple Actions.
Order	Enter an integer to indicate the order in which the actions must take effect in the wizard page. ***Tip:*** *When defining the order, you may want to skip a few numbers before each action (for example, 3, 6, 9, 13, 18, 22, and so on) in case you may need to insert a few additional actions later, for example, if your design changes.*
Object	Select the object for which you want to define a simple action. The objects displayed in this drop-down list include:

	• The object for which wizard is being created, for example, Claim. • All sub-objects or related objects defined in the template, if you are using one with the wizard. These objects are numbered the same way as they are in their respective template screens, for example, Assignee 1, Assignee 2, Category 1, Task 1, Task 2, and so on.
Attribute	Select the attribute of the field in the user interface to which you want to assign a value.
Operator	Select the appropriate operator, depending on the operation you want to perform on the attribute value. For examples, see the Operator Options for Different Attribute Types table. ***Note:*** *Along with the operators in the Operator Options for Different Attribute Types table, there is a set of the same operators available. These operators use the value of a* ***Parameter*** *instead of* ***Path*** *or* ***X****, for example,* ***Set to Parameter****,* ***Set to Value of Parameter****,* ***Subtract X from Value of Parameter****, and so on.*
Path	This field and the **reset** button appear when you select a dynamic or static-dynamic option from the **Operator** drop-down list. For examples of the operators, see the Operator Options for Different Attribute Types table. Click the **Object Navigator** icon next to the field and select the appropriate path in the displayed Object Navigator window. For details on how to use Object Navigator, see Using Object Navigator.
reset	Click to clear the **Path** field and start adding the path again.
add more	Click to add the new item to the list of page actions.
Page Action List	Displays all simple actions added to the selected wizard page. Select the check-boxes next to the actions that you want to delete or edit and click the appropriate command button at the bottom.

The following table lists possible options for different attribute types and examples of how they might be used.

Operator Options for Different Attribute Types

Attribute type	Operator	Value type	Description	Example

Lookup Table **Check-Box** **Object** **IID**	Set to X	Static	Assigns the value entered or selected in the **X** field.	The **Main Assignee** for the claim is the assignee with the role **Adjuster**.
	Set to value of path	Dynamic	Assigns the value of the path specified in the **Path** field. ***Note:*** *For check-boxes, the value of path may be another check-box. For lookup tables, the value of path must be the same lookup table in another record.*	None
Date	Set to X	Static	Assigns the value entered in the **X** field.	The **Due On** date of a task is set to the date entered in the **X** field, for example, '07/01/02'.
	Set to value of path	Dynamic	Assigns the value of the path specified in the **Path** field.	A task is due the same day as the date the claim is created. If the claim is created on 07/01/02, the **Due On** date for the task is set to 07/01/02.
	Add X days to value of path	Static-Dynamic	Adds the number of days entered in **X** to the value of the path.	The task '30-Day Review' is due 30 days after the date the claim is created. If the claim is created on 07/01/02, the **Due On** date for the, '30-Day Review' task is set to 07/31/02.
	Subtract X days from value of path	Static-Dynamic	Subtracts the number of days entered in **X** from the value of the path.	The Progress Report for a claim must be printed 2 days before the, '30-Day Review' meeting.

				If the, '30-Day Review' meeting is scheduled for 07/31/02, the due date for the Progress Report is 01/29/01.
Text (Strings)	Set to X	Static	Assigns the value entered in the **X** field.	The value of the **Short Description** field for a task is set to the string entered in the **X** field, for example, '30-Day Review'.
	Set to value of path	Dynamic	Assigns the value of the path specified in the **Path** field.	The value of the **Name** field of the account is set to the name of the claim. For example, 'Allison McDonald Auto Accident'.
	Concatenate X before value of path	Static-Dynamic	Include the value entered in **X** before the value from the **Path**.	The value of the **Short Description** field for a task is set to the string `30-day review for' entered in the X field + the name of the claim. For example, '30- day review for Allison McDonald Auto Accident'.
	Concatenate X after value of path	Static-Dynamic	Include the value entered in **X** after the value from the **Path**.	The value of the **Short Description** field for a task is set to the name of the claim + the string `-30-day review' entered in the **X** field. For example, `Allison McDonald Auto Accident - 30- day review'.
Number	Set to X	Static	Assigns the value entered in the **X** field.	The value of the **Allocation Limit** field of an account is set to the number entered in the **X** field, for example, `1,000,000.00'.
	Set to value of path	Dynamic	Assigns the value of the path specified in the **Path** field.	The value of the **Allocation Limit** field of an account is equal to that of the **Allocation Limit** of its parent account.

				If the Allocation Limit of the parent account is set to `250,000', then the Allocation Limit of the account is set to `250,000'.
	Add X to value of path	Static-Dynamic	Adds the number entered in **X** and to the value of the path.	The value of the **Allocation Limit** field of an account is the sum of `50,000' and the **Allocation Limit** of the parent account. If the Allocation Limit of the parent account is set to `250,000', then the Allocation Limit of the account is set to `300,000'.
	Subtract X from value of path	Static-Dynamic	Subtracts the number entered in **X** from the value of the path.	The value of the **Allocation Limit** field of an account is the difference of `50,000' and the **Allocation Limit** of the parent account. If the Allocation Limit of the parent account is set to `250,000', then the Allocation Limit of the account is set to `200,000'.
	Multiply the value of path by X	Static-Dynamic	Multiplies the value of the path by the number entered in **X**.	The value of the **Allocation Limit** field of an account is 25% of the **Allocation Limit** of the parent account. If the Allocation Limit of the parent account is set to `250,000', then the Allocation Limit of the account is set to `62,500'.
	Divide X by the value of path	Static-Dynamic	Divides the number entered in **X** by the value of the path.	None

Number	Divide the value of path by X	Static-Dynamic	Divides the value of the path by the number entered in **X**.	The value of the **Allocation Limit** field of an account is half of the Allocation Limit of the parent account. If the **Allocation Limit** of the parent account is set to `250,000', then the Allocation Limit of the account is set to `125,000'.
User	Set to X	Static	Assigns the value selected in the **X** field.	The value of the **Assignee** in a claim is set to the user selected in the **X** field, for example, Jane Smeeth.
	Set to value of path	Dynamic	Assigns the value of the path specified in the **Path** field.	The value of the **Assignee** field in a claim is set to the user who created the claim. If Jane Smeeth is a claim associate who created the new claim, she is automatically set as the assignee for the claim.
	Set to user with role X in project of path	Dynamic	Assigns the user with the specified role in another project.	The value of the **Assignee** field in a claim is set to the user who is specified as the **Underwriter** for its parent policy. If Mathew Anderson is the Underwriter for the parent policy, he is automatically set as the assignee for the claim.

1.1.11.2.6 Defining Page Transition Logic

Page transition logic is the sequence in which the wizard pages follow each other. Depending on user input, this sequence may be linear (all wizard pages following each other), may skip pages, or may branch into alternative flows.

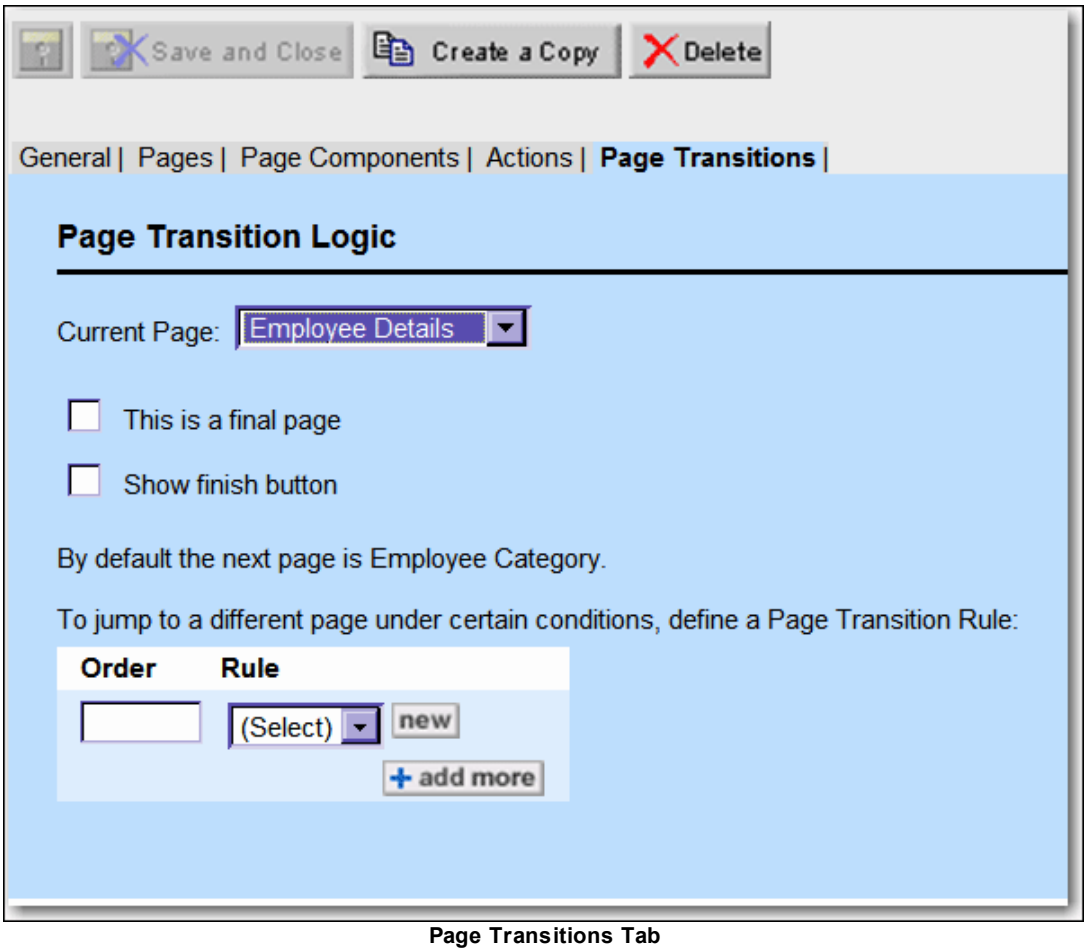
You can also allow users to end a wizard on any page by adding a **Finish** button. This allows users to choose where they want to stop the wizard, and then use the record's additional pages to add more information at a later time.

You must plan your page transition logic and create the wizard pages before specifying the necessary page transition rules. By default, the **Order** numbers you assign to each page defines the sequence of pages in a wizard.

To define page transition logic

Note: *This procedure assumes that the wizard is open to its tabbed pages.*

1. Click the **Page Transitions** tab for the appropriate wizard.



Page Transitions Tab

2. From the **Current Page** drop-down list, select the page for which you want to define the transition logic.
3. Complete the following information:
   - If the selected page is the last page of the wizard, click the **This is a final page** check-box.

Caution: *If you select the* ***This is a final page*** *check-box, the* ***Create the record on finish*** *check-box appears. If you do not check the* ***Create the record on finish*** *check-box, when you reach the end of the wizard, all of the information may be discarded.*

- If you want to add a **Finish** button to the selected wizard page, click the **Show finish button** check-box.

4. In the **Order** field, enter a number to indicate the order in which the rule must take effect in the wizard page.
5. Select one of the following options:
   - To select an existing rule, click the appropriate rule in the **Rule** drop-down list.

     You may view the details of a rule by selecting it and clicking **Open**, which displays the Rule screen where the qualifier and action are defined.
   - To create a new rule, see Creating Page Transition Rules.

     The newly created rule is now displayed in the **Rule** drop-down list.
6. Click **add more**.

   The selected rule is added to the list of rules defined for the page.
7. Repeat steps 2 through 6 to add other rules to the page, if necessary, and then click **Save**.

To display another wizard after a user completes the first wizard

Note: *You can only add another wizard to wizards for custom objects.*

1. From the **Page Transition** tab of a wizard, place a checkmark in the **This is a final page** checkbox.

   The **Pick Wizard to launch on finish** options appear.
2. Select the **Object Definition** with the second wizard.
3. Select the wizard name from the **Wizard** drop-down list.

The following table describes the items on the **Page Transitions** tab.

Page Transitions Tab

Field or button	Description
Current Page	Select the wizard page for which you want to view or define page transitions. ***Note:*** *All pages in this field are listed in the order assigned to them on the* ***Pages*** *tab.*

This is a final page	Select this check-box if the selected page is the last page that users will see, regardless of the flow that they select. The screen refreshes to display the **Create the record on finish** check-box and the **Post Wizard to launch on finish** options.
Show finish button	Select this check-box if you want to add a **Finish** button to the selected page of the wizard. You can add a **Finish** button to any page of the wizard.
Create the record on finish	Select this check-box if you want to create the record when the user clicks the **Finish** button. If you clear the check-box, all the information collected throughout the wizard may be discarded.
Order	Enter an integer to indicate the order in which the page transition rule takes effect in the selected wizard page.
Rule	Select the page transition rule you want to apply to the selected wizard page.
new	Click to create a new page transition rule for the selected page.
add more	Click to add the selected rule to the page transition rule list at the bottom of the screen.
Pick Wizard to launch on finish	If you want another wizard to display after the user completes this wizard, select the following options: • **Object Definition**—The object with the wizard that you want to open. • **Wizard**—The wizard that you want to open after the user completes the first wizard. These options only appear when you select the **This is a final page** check-box.

1.1.11.2.6.1 Creating Page Transition Rules

Page transition rules allow you to specify alternative flows in response to user input and other qualifiers, so the wizard displays the appropriate pages to the user. Otherwise, all pages follow each other in the order specified on the wizard's **Pages** tab.

Page transition rules are similar to TeamConnect rules located on the **Rules** tab of an object definition screen. However, page transition rules are wizard-specific, so you may only create and access them from the **Page Transition** tab of the corresponding wizard screen.

Page transition rules have the following tabs:

- **General**—General information about the rule, including its name, whether it is active, and its trigger. The rule's **Type** field has the value **Wizard Page Transition**, which cannot be changed.
- **Qualifier**—Allows you to define the qualifier that must be met in order for the rule to take effect. Wizard parameters may serve as qualifier items.
- **Action**—Allows you define which page comes next, if a specified condition exists.

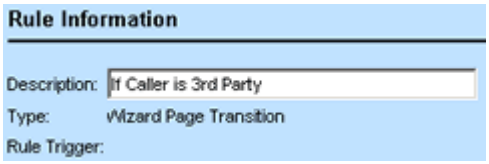
When the user clicks **next**, page transition rules display the appropriate page based on the specified conditions. Like validation rules, they may check the values entered in certain fields, check who the user is, and so on.

Wizard parameters that you add on the **Page Components** tab of the wizard screen become available on the **Qualifier** tab of the page transition rule screen. That way, user entries in the wizard may be used as qualifiers in the rule.

To create a page transition rule

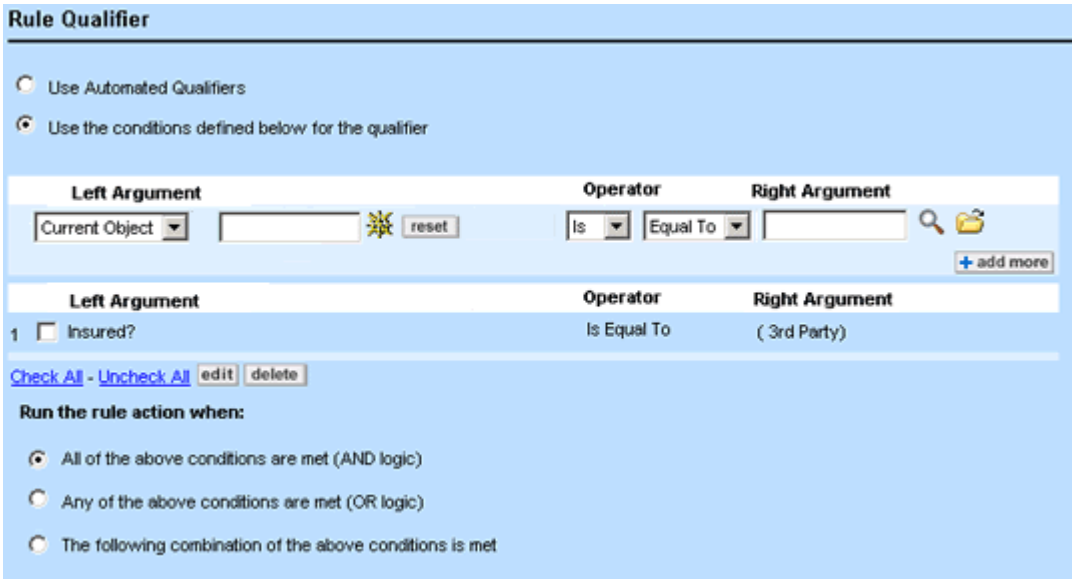
1. Open the **Page Transitions** tab of the appropriate wizard.
2. In the **Current Page** drop-down list select the wizard page for which you want to define the rule.
3. In the **rule** column click **New**.

   The **General** tab of the wizard page transition rule screen opens.



General Tab in Wizard Page Transition Rules

4. In the **Description** field, enter a self-explanatory name for the rule.
5. Click **Save** to allow you to add qualifiers.
6. Select the **Qualifier** tab.



Qualifier Tab in Wizard Page Transition Rules

7. Select the appropriate **Left Argument**, **Operator**, and **Right Argument** for the rule you are creating.

   The **Left Argument** list includes all parameters that have been added to the wizard.

8. Click **add more**.

9. Select the **Action** tab.

Action Tab in Wizard Page Transition Rules

10. From the **Next Page When Condition Verified** field, select the page that is next if the qualifiers are met.

11. Click **Save**.

    The rule is now available in the **Rule** list on the **Page Transitions** tab.

Points To Remember

Keep in mind the following important information about page transition rules:

- Page transition rules are executed on a per-page basis—that is, when the user clicks **next** in the wizard, not on Create when the user clicks **finish**.
- The **Action** tab of a page transition rule allows you to specify the page to which the wizard must direct the user, if the conditions on the **Qualifier** tab is verified.
- When a user clicks **next** in a wizard page for which a rule is defined, page transition rules are executed on a per-page basis according to the **Order** numbers you assign to each rule.
- Page transition rules are wizard-specific and may be created and accessed only from the **Wizard** screen of the object for which the wizard is created, not from the object's **Rules** tab.

- To direct a user to a specific page depending on the rights of their user group, check the current user in the page transition rule. The **Current User** option is available from the **Left Argument** drop-down list on the **Qualifiers** tab.
- You may add wizard parameters as page transition qualifiers on the **Qualifier** tab. Select them from the **Left Argument** drop-down list. Parameters are also available when you select **Attribute** from the **Right Argument** list.

Example: Left Argument Drop-down List

Page Transition Rule Example

Assume there are five pages in a wizard regarding view obstructions that a claimant and insured might have had. The user must provide the following information in each subsequent page depending on the answers:

Page Transition Rule Example

Page #	Wizard question	Field type	Wizard page component	Page transition
1	Did the Insured have any view obstructions?	Yes or No Radio buttons	Parameter1	If yes, go to Page 2. If no, go to Page 3.
2	Indicate the view Obstructions that the Insured had.	Check-Boxes	Block	Go to Page 3.
3	Did the Claimant have any view obstructions?	Yes or No Radio buttons	Parameter2	If yes, go to Page 4. If no and yes on Page 1, go to Page 5. If no and no on Page 1, go to Page 6.
4	Indicate the view Obstructions that the Claimant had.	Check-Boxes	Block	Go to Page 5.

5	Provide a summary of the view obstructions that the Insured and Claimant had.	Memo Text	Custom Field	Go to Page 6.
6	Did the Insured receive any citations?	Yes or No Radio buttons	Parameter3	

For example, to create a rule to skip to Page 6 from Page 3 when neither Insured nor Claimant had any view obstructions, you have to do the following actions:

1. On the **Qualifier** tab of the rule, you must specify the following conditions:
   a. Select Parameter1 in the **Left Argument** drop-down list and set it equal to **No**.
   b. Select Parameter2 in the **Left Argument** drop-down list and set it equal to **No**.
2. On the **Action** tab of the rule, you must select Page 6.

1.1.11.2.6.2 Checking Page Order in Wizards

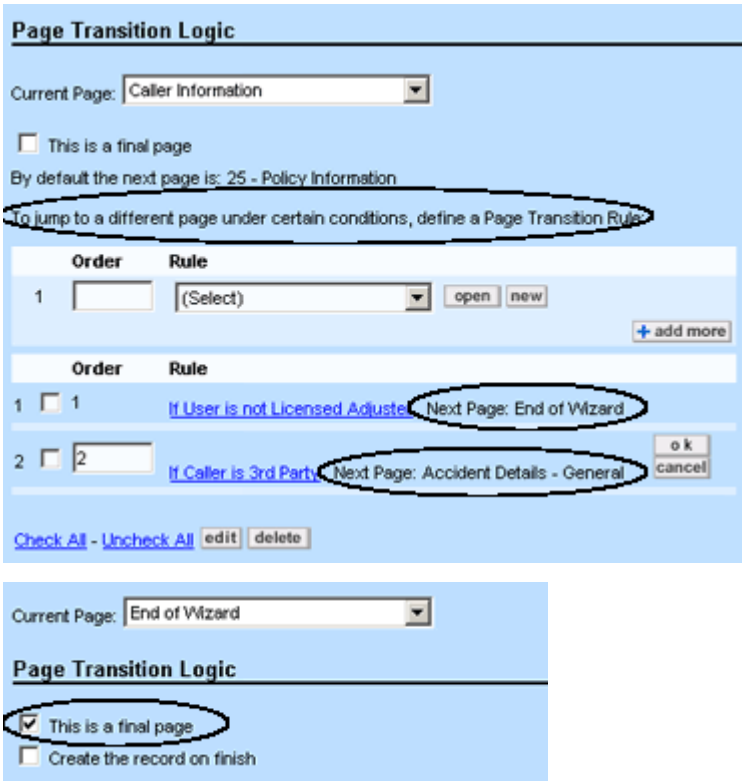
To determine the page order, alternate flows, and final pages in a wizard, refer to the **Page Transitions** tab of the corresponding wizard screen.

The **Page Transitions** tab does not list all page transitions at once. Instead, you select each wizard page in the **Current Page** drop-down list to determine which page follows it, based on the default page transition order or the selected page transition rules.

To check the page order in a wizard

1. Open the **Page Transitions** tab in the appropriate wizard.
2. From the **Current Page** drop-down list, select the wizard page for which you need to check the page order.

   The screen either displays the page that follows the selected page, or informs you that the selected page is a final one, as shown in the following images.



Page Transitions Tab Screens

3. Repeat the previous step for all wizard pages you need to know the transition order for and write down the page transition sequence for each.
4. Make any necessary changes and close the wizard screen when you are done.

You may use the list of page transitions to help you avoid breaking wizard flows when making changes to the page transitions.

1.1.11.3 Testing and Troubleshooting Wizards

After you have designed a wizard, you must thoroughly test it and resolve any problems that may arise.

To test a new wizard

1. Make sure you have the rights to the object for which you have created the wizard.
2. From the end-user interface, click **New**.

Select a Wizard Pop-up Window

3. Click the name of the wizard you are testing, in the pull-down menu.
4. Go from page to page selecting different options and checking where each option leads you.
5. Click **Finish**.
6. Repeat steps 4 and 5 for each of the alternative flows in your wizard, if applicable.
7. Make sure the wizard works as designed by checking:
   - The page order
   - Page transition rules' operation
   - The record state when you click **Finish** (record saved correctly)
   - Whether values are populated as assigned
   - Related objects specified in the linked template, making sure they are created and populated appropriately
   - Whether the wizard has correct layout, typographical errors, and field-and-block alignment
8. Make the necessary changes to fix any problems and test the wizard again.

The following table lists common problems and possible solutions to them.

Wizard Troubleshooting Tips

Possible Problems	Possible Solutions
The **New** button does not appear at all on the TeamConnect user interface.	The **Require wizard for new object definition** check-box is selected, but there are no active wizards. Either clear this check-box or activate a wizard. ***Note:*** *This does not apply to Contacts—a* ***New*** *button is always visible because it is also used to create Address Books.*
The wizard is not displayed in the New drop-down list.	Make sure the wizard is activated by selecting the **This wizard is active** check-box on the General

		tab of the wizard.
I want users to only be able to create records by using a wizard, but when I click the **New** drop-down list, I still see the wizard option and the blank record option.		Select the Require wizard for new object definition check-box on the **General** tab for the wizard. See the General Tab on Wizard Screens table for more information.
The selected wizard cannot be started.		• Make sure the correct automated action file is selected as the initial action for the wizard. • Ask a developer to check the code in the initial action file.
The wizard never ends.		Make sure the page transition rules are added properly, and no loops are created by switching the same pages over and over again.
Page order is not correct.	Pages do not follow each other in the order that was assigned to them on the **Pages** tab.	Make sure there are no page transition rules added to your wizard pages on the **Page Transitions** tab, because this action modifies your default order.
		Make sure there are no Final pages added in the middle of your default page order.
	Certain pages never display.	Make sure they have not been accidentally deleted from the wizard.
		Make sure the page transition rules are added correctly. All possible options for the user input are accounted for. No loops are created.
		Make sure there are no final pages added in the middle of your default page order.
		Make sure no pages with the added page transition rules have been deleted, thus breaking the sequence.
	Pages are not skipped as appropriate or appear when they are supposed to be skipped.	Make sure the qualifiers are set correctly in your page transition rules. All possible options for the user input are accounted for.
	Transition rules use the wrong values.	Make sure the parameter and custom field names in custom blocks are unique.

	The wizard goes to the final page without having gathered all the necessary information.	Make sure you use the appropriate user account to test your wizard. For example, if you are an Administrator, and the wizard is set to allow creating records only to the users with the role Claim Associate or Claim Adjuster, you are not able to proceed.
		Make sure the required fields are populated.
Records are not created.	The wizard does not allow you to save the record when clicking **Finish**.	Required fields may not have been populated or were not added to the wizard. For example, a parent record, or a custom field that are included in the object views.
	The wizard exited but the record was not created.	Make sure the **Create the record on finish** check-box is selected on the **Page Transitions** tab of the Final pages appropriately.
	The wizard was supposed to end without creating the record, but the record was created.	
	No related records were created when the wizard finished.	Make sure the appropriate template is linked to the wizard in its **General** tab.
		Make sure the appropriate related objects are added to the template linked to the wizard.
The assigned default values are not populated correctly in the saved record.		Make sure the correct attributes or paths were selected for the fields in the template or on the **Actions** tab of the wizard.
		Make sure the values assigned in the template are not overwritten on the Actions tab of the wizard, by the initial action, or manually when working with the wizard.
Layout and Formatting	The fields and their labels are not aligned.	Consider putting individual fields into a separate custom block.
	The fields and their values are not formatted appropriately, for example, to display time, dollar amounts, and so on.	Create custom blocks for the custom fields and add the appropriate formatting tag attributes.

		Tip: *You may add basic HTML formatting to the text components, such as* ***bold****, italics, underline.*
	The last page in the wizard does not have the finish button.	Make sure it is set as a final page on the **Page Transitions** tab.
	Duplicate parameter fields appear on the same page.	If you have included parameters in a custom block, make sure the **Display in blocks only** check-box (see the Select a Wizard Pop-up Window image) is selected on the page where the parameter is defined.
	Parameters do not appear on the page.	If the parameters are not included in custom blocks, make sure the **Display in blocks only** check-box (see the Select a Wizard Pop-up Window image) is not selected. If the parameters are included in custom blocks: • Make sure the block is added to the page where the parameters are defined or on any other page that follows. • Check whether the parameters are within a `<tc:section>` tag that contains custom fields in the custom block. If they are, either make sure the category to which the custom fields belong is added to the record that is being created through the wizard, or move the parameters outside a `<tc:section>` tag (see Sample XML File).
Custom Blocks	Blocks take long to load.	The first time you are using your wizard, it may take a little longer to load the custom blocks, because they need to be cached locally.
	Block does not appear on the wizard page. There is an error instead.	The XML file may have problems. Make sure the custom block was created properly. For details, see: • Screen Designer Help • Troubleshooting Custom Blocks
Miscellaneous error messages may appear during the testing phase.		Make sure the fields used in actions or page transition rules are populated.

1.1.11.3.1 Execution and Population Order

To better understand how templates and wizards work and to be able to troubleshoot them, it is important to understand the order in which values are populated and actions are executed.

Starting a Wizard

When a user starts a wizard the following actions happen, in this order:

1. All values from the associated template are automatically populated.

   The sub-objects and related objects specified in the template are created and the default values are assigned to the fields of the main record.

2. The automated action selected as the initial action for the wizard is executed.

   If there is no initial action specified, this step is skipped.

3. The parameters defined on the first page of the wizard are created.
4. The first page of the wizard opens with the latest values of all objects and parameters.
5. When the user clicks **next**, the available page actions and page transition rules associated with the page are executed.
6. The next wizard page opens with the updated values.

1.1.12 Using Rules

Rules allow you to automate your organization's business requirements within TeamConnect. Rules prevent records from being processed incorrectly or without proper approval by performing validations, security checks, and approval processes that control the flow of data. Rules eliminate the need for users to perform such checks and create a more reliable workflow.

TeamConnect contains many rules already, to coordinate workflow among and between its objects. As you add more types of objects and implement more capabilities, you may need to create additional rules.

Creating rules is one of the last steps in the customization process (See Customization Sequence). Before starting to create rules, make sure you have finished the following operations:

- Defining custom objects
- Adding necessary items to system lookup tables
- Defining object categories
- Creating custom fields and custom lookup tables
- Organizing custom fields by creating blocks
- Setting object views and search views
- Creating group accounts with the appropriate rights

1.1.12.1 Rule Components

Rule definitions include the following components:

- Rule Types—Select the appropriate rule type and save the rule before defining other components.
- Rule Triggers—The end-user action that triggers the rule, such as creating a record.
- Rule Qualifiers—The information that the rule checks before executing its action. A rule qualifier consists of a list of qualifier items and the logic used to evaluate those items.
- Rule Actions—The action that the rule performs when its qualifiers are met, such as denying the operation attempted by the user or sending it for approval. More complex actions, such as generating files when the rule's qualifiers are met, are considered custom actions and may be accomplished by writing rules using class files or JavaScript. The pre-population of fields may be defined through the use of templates.

1.1.12.2 Rule Types

You may create several different types of rules in TeamConnect:

- Security Rules—Protect records when they are in certain states. For example, when users open a matter record that is in a certain phase, no one may be able to modify it except for a certain group of users.
- Pre-population Rules—Populate fields with predefined values. For example, a filing date field may be populated when the **Lawsuit** category is added to a dispute record. You may define pre-population rules in conjunction with templates, which you may create through Designer without the need for class files or JavaScript.
- Validation Rules—Confirm the conditions that exist in a record at the time an action is attempted. For example, when a user clicks save, determine whether the appropriate custom fields are filled out for each category. If not, prompt the user with a message and do not save the changes.
- Approval Rules—Send requests for approval to specified users before an action is completed, such as posting an invoice. The requests then appear in each of those users' **My Approvals** screen.
- Custom Action Rules—Perform custom actions according to the needs of your organization. These actions are defined in class files or JavaScript or through the use of a template.
- Scheduled Action Rules—Custom action rules with parameters set through Designer to schedule when and how often an action is executed. Unlike custom action rules, scheduled actions do not support templates.
- Audit Rules—Capture and log events by creating history records that contain information about user activities in TeamConnect.
- Post Commit Rules—Perform a task after an action has completed successfully. For example, send an email notification to associated users after a record has been updated.

These types of rules may be categorized into the following groups based on the actions that they may perform:

- **Predefined actions**—Defined through Designer, security, pre-population, validation, approval, and audit rules have predefined actions such as the following:

- Deny an operation.
- Send the operation on a route for approval.
- Populate fields based on values in a template.
- Record user activities in history records.

- **Custom and scheduled actions—**Defined by creating class files or JavaScript and the TeamConnect API, custom action rules may perform any action written in Java or JavaScript.

  Scheduled action rules use custom Java or JavaScript actions that you may schedule using certain parameters in Designer.

  Custom action rules also support templates, whereas scheduled action rules do not. However, the predefined actions defined by a template are not custom actions.

  By default, TeamConnect does not perform duplicate contact checking if custom code creates or updates contacts. See the `ContactService.readDuplicateContacts()` method in the API Javadocs to use duplicate contact checking functionality.

- **Page transitions—**Defined on wizard screens, you may create rules to govern how users enter information in a wizard. For more details about page transition rules, see Creating Page Transition Rules.

1.1.12.2.1 Security Rules

Security rules prevent an operation (such as creating, updating, or deleting) from being completed under specified circumstances and based on the original values of the record when it was opened by the user. When the user tries to alter a record, if the original values match the qualifier specified in the rule, the rule prevents the operation and displays a message to the user.

Keep the following points in mind when working with security rules:

- When a user attempts an operation, TeamConnect executes any applicable security rules before executing other types of rules.
- Security rules compare the rule qualifier to the conditions of the record when the user first opened it, so they prevent a user from changing values in the record to get around the rule. Qualifiers such as **isChanged**, **Item Deleted**, **Item Modified**, or **Item Added** will not have any effect on the rule's actions, since the rule is checked before any changes have occurred.
- Security rules do not provide field-level security. For example, you cannot specify that a certain field's value may not be changed. Security rules may only prevent the user from updating or deleting the entire record based on certain specific, pre-existing conditions in the record.
- You may use security rules to control the Delete and Update operations for existing records belonging to any object, or for the Phase Change operation in a custom object. For a complete list of operations that may trigger security rules, see the Rule Triggers table.
- You may define security rules for specific user accounts and user groups. For example, if the current user is not a member of the Adjusters group, that user cannot update the record while it is in a specified phase.

1.1.12.2.2 Pre-population Rules

Pre-population rules may populate fields, sub-objects, or related objects with values predefined in a template, which you may access or create on the **Templates** tab of the object definition. Once you create a template, you may select it from the **Action** tab of your pre-population rule screen.

Note: *In addition to pre-population rules, you may use templates with custom action rules and wizards.*

With templates, you may create rules that:

- Populate fields of existing records
- Create a new record and populate its fields
- Create a related record for the current object, and populate its fields
- Populate assignees and attendees and specify the main assignee by role
- Perform phase changes
- Populate the default category of a record
- Specify the default category with a static value

The available triggers and qualifiers for pre-population rules are the same as validation rules. For a complete list of triggering operations, see the Rule Triggers table.

For information about defining templates for object definitions, see Using Templates.

1.1.12.2.3 Validation Rules

Based on the values in the record, *validation rules* prevent users from creating or updating the records, or changing project phases. If the qualifier in the rule is met, the rule prevents the operation and displays a message to the user.

Points To Remember

- When a user attempts an operation, validation rules execute after all applicable security rules and pre-population rules, as described in the execution order listed in Points to Remember.
- Validation rules compare their qualifiers to the values in the record at the time when the user attempts an operation, not when the record was opened.
- Validation rules may control the Create or Update operations for any object, or for the Phase Change operation in a custom object. See Rule Triggers for a complete set of triggers according to each object.
- Page Transition rules in wizards function like validation rules. They "validate" the information that the user enters in each wizard page to ensure that the data is entered properly. They also use this information to determine which wizard page to display next. For more information, see Creating Wizards.

Validation Rule Example

For example, you might write a validation rule for involved party records of the Dispute object, which checks:

- Whether the current user who is attempting to update a dispute record is a member of the Dispute Attorney group
- Whether the involved party record has the **Outside Counsel Firm** category
- Whether the **Involvement Date** field of the involved party record is empty (has a null value)

If all of these conditions are present, the rule prevents the record from being updated and displays a message to the user such as **Involvement date of the involved party cannot be empty**.

This is a validation rule rather than a security rule because the value in the **Involvement Date** field when the user opened the record is irrelevant. The purpose of the rule is to make sure that the **Involvement Date** field has a value when it is saved.

Complementary Validation Rules

When writing a validation rule for the Create operation, make sure that you also write a validation rule for the Update operation that complements the Create rule, if necessary. This action ensures that the conditions you specify are met at all times.

For example, if you want to ensure that dispute records are always saved with at least one assignee, you must create two identical validation rules: one for Create and one for Update. If only the Create rule is used, a user may open an existing record and save (update) it after that user has deleted the assignee. To prevent this action, write a rule that requires an assignee when users update the record.

1.1.12.2.4 Approval Rules

Approval rules designate certain users who must approve an operation before it may be completed. For example, you may define approval rules to require permission to post invoices, deactivate accounts, change the phase of a matter. The Rule Triggers table lists the available operations for approval rules.

Use approval rules in conjunction with routes to define approval processes. A Route specifies a logical sequence of approvers used in an approval process. These approvers are included in routes through their user group memberships, since routes are based on groups, not individual users. For more details about routes, see Creating Routes.

TeamConnect performs the following operations when an approval rule is triggered:

- When a user attempts a triggering operation, TeamConnect executes applicable approval rules after all security and validation rules for the operation are executed, as described in the execution order listed in Points to Remember.

  For example, if there is a security rule preventing users from deleting a record when it is in a certain phase, this rule is executed before the approval rule sends the deletion for approval.
- If the qualifier defined in the approval rule is met, the operation (also known as a request) is put into a pending status.
- The specified message appears at the top of the record, such as **This object has an action that is pending approval and it cannot be modified**.

- The approval request goes through the specified route of approvers. As each approver receives the request, it appears on his or her **My Approvals** screen.
- If the required users approve the operation, TeamConnect completes the requested operation.
- If the required users reject the operation, TeamConnect does not do the requested operation.
- During the approval process, decisions are listed on the **Workflow** tab of the record. If the needed approval is for deleting the record, the decisions cannot be accessed after the record is deleted.

You may create qualifiers for approval rules so that the user's attempt to perform an operation only goes for approval under certain conditions. For example, you may create an approval rule that requires approval for deleting a closed dispute record only if the deletion is attempted by a user in the Dispute Attorney group.

You may also specify an approval period for approval rules. You must use a value other than zero when specifying the length of an approval process. Otherwise, the approval period expires almost immediately.

If you do not set any qualifier information for an approval rule, the operation requires approval under all circumstances.

If an approval rule is triggered by the only user who is included in the route, the rule still requires that user's approval. The workflow status message immediately appears at the top of the record and the user must approve the operation before it is completed.

Many other actions and events may take place during an approval process.

1.1.12.2.5 Custom Action Rules

You may write custom action rules in Java or JavaScript and use Designer to upload the files to TeamConnect and select them for use in the rule. The qualifier for a custom action rule may be defined in Designer, or it may be a custom qualifier created in Java or JavaScript (an automated qualifier). For details, see Writing Automated Actions.

You may define custom actions using a template, which you may access or create on the **Templates** tab of the object definition. You may select the template from the **Action** tab of your custom action rule.

Note: *By default, duplicate contact checking is not done when custom Java code creates or updates contacts. See the ContactService.readDuplicateContacts() method in the API Javadocs to use duplicate contact checking functionality.*

Custom Action Rule Example

Custom action rules define actions that cannot be specified in Designer. For example, you might write a custom action rule in Java to automatically create budgets when a user adds an outside counsel firm to a dispute record. You upload the class file to the appropriate folder in TeamConnect's Documents area and then select it from the list of class files on the rule's **Action** tab.

1.1.12.2.6 Scheduled Action Rules

A *scheduled action rule* is similar to a custom action rule. You upload custom Java or JavaScript files to use for the rule action. The difference is that when you define these rules in the user interface, you have the option to specify when and how often the action is executed.

For example, your system might need a custom rule requiring, when the qualifiers are met, a task is created every 15 days to remind the user to do something, for a period of 90 days. To accomplish this requirement, a custom Java class might be uploaded to TeamConnect, and you then select this class file in the scheduled action rule. However, the actual number of days between tasks and the number of times that a task must be created are specified on the **Action** tab of the scheduled action rule.

When a scheduled action rule is triggered, you may view information about actions pending in the Scheduled Actions Monitor, provided that you have the rights to this tool. If necessary, you may also delete (or cancel) pending actions using the Monitor screen.

If the qualifying conditions for a scheduled action rule are true, and the rule is triggered, it will continue to run as scheduled even if the qualifying conditions subsequently become false during the schedule. The qualifiers are only checked the first time the scheduled action rule is invoked.

For more details about the Scheduled Actions Monitor and scheduled action rules, see:

- Scheduled Actions Monitor
- Creating Automated Qualifiers and Actions
- Defining Settings for Scheduled Action Rules

By default, duplicate contact checking is not done when custom Java code has been written to create or update contacts. See the `ContactService.readDuplicateContacts()` method in the API Javadocs to use duplicate contact checking functionality.

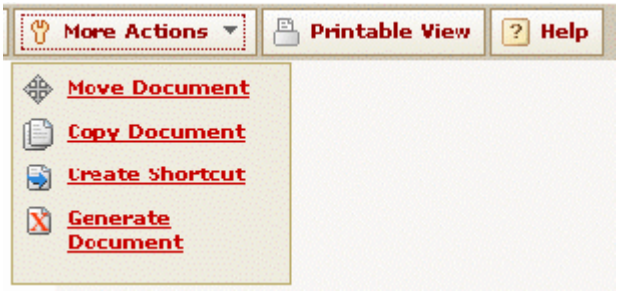
1.1.12.2.7 User Invoked Actions

You may define custom actions to be user invoked. Instead of the rule being triggered by an action like the creation or deletion of a record, the user intentionally invokes the action by selecting it from a drop-down list. The qualifier items are then checked and the action is executed if the qualifiers are met.

For example, you could define a user invoked custom action to trigger an email message to be sent containing information from the current record.

You may develop user invoked custom actions to create child records. To do so, you must specify whether the record (and the child record to be created) may be created when the user invoked rule is performed or when the user clicks **Save**. Either alternative is possible.

The drop-down list for user invoked actions appears to users on the toolbar, as a **More Actions** button.



Example: User Invoked Actions Drop-Down List

To create a user invoked custom action rule

1. Define the custom action as a Java class or JavaScript file.
2. Define a custom action rule for the appropriate object definition and set the rule trigger to **User Invoke**.

   When you define a user invoked action, the user invoked action becomes available on the **Rights** tab of user and group accounts for the corresponding object definition.
3. Give rights to all groups to whom you want to give access to the custom action.

   The next time they log in, members of those groups have the action available to them on the record toolbar.

1.1.12.2.8 Audit Rules

Audit rules may keep track of activity in TeamConnect by logging information in history records. You may apply these rules to user actions that are relevant for record keeping or monitoring changes, such as when a user updates one or more system or custom fields in a record. Audit rules provide you with a way to define business rules through Designer rather than developing and testing custom code to record user activity.

Audit rules may create history records that include the name of the user performing an action and how the record was affected. For example, you may create an audit rule to track when a contact record's **Social Security Number** field is updated. When the rule is triggered, it records the old value, new value, and the user who made the change.

Recording information in history records allows you to control what is recorded and how it appears. For example, you may create a custom description for the event that triggers the audit rule, record information in custom fields, and export data for reporting purposes.

Points To Remember

Keep the following points in mind when creating audit rules:

- You may use audit rules to track modifications to data, and the creation or deletion of records.
- When an audit rule is triggered and its qualifiers are met, it creates one history record. To define audit rules that are triggered on Update, define qualifiers that only capture information when appropriate. Otherwise, a history record will be created whenever a user updates any part of a record.

- Rules with multiple triggers have the potential to execute more than once for a single event. If this happens with an audit rule, then multiple history records will be created.
- TeamConnect executes audit rules after all other business rules are applied. That way, operations that are prevented because of another rule will not cause the creation of an audit history record.
- The current user's access rights do not affect the creation of an audit history record. For example, even if the current user does not have rights to create related records in matters, the audit rule may still create a history record for the current matter record.
- If an audit rule is triggered but a history record cannot be created, for whatever reason, no additional information is logged, and no error is given to the user.

For information about the specific settings available in audit rules, see Defining Actions for Audit Rules.

1.1.12.2.8.1 Updates, Triggers, and Audit Rules

If you choose "use default description" in the Actions tab of an Audit rule, TeamConnect autopopulates the "Text" field in the audit history record. The way in which "Text" is autopopulated differs depending on the nature of the trigger.

For example, the text differs depending on whether an object is being created, updated, or deleted. Here is an example of the "Text" field in an audit history record that is triggered by an Update of a Project object:

[When the action was completed the following information was changed in Matter 'test']

- **Amount was '10,000' and was changed to '15,000'**
- **Transaction Date was '01/01/05' and was changed to '01/02/06'**
- **Quantity was '25' and was changed to '50'**

The default description for an update contains an entry for each field that changed during the update (three fields in this example). However, if you choose a trigger other than "Update", you may not get as much information.

For example, the Account object allows you to define an audit rule that uses the "Withdraw" trigger. The resulting audit history record tells you that a withdrawal took place, but does not specify the actual amount of the withdrawal, whereas an audit history record based on the "Update" trigger would show the change in amount.

You must decide how much detail is required and choose whether one trigger is sufficient or whether you need to associate both triggers with the rule. Here is a more complete syntax for how TeamConnect constructs the default description for an Update trigger:

```
[When the action was completed the following information was changed in <Record>]
- <Field> was <Old Value> and was changed to <New Value> [For non memo text fields]
- <Field> was changed [For memo text fields]
[List of the changed, added or removed sub-objects and their associated events and fields in separate lines. Note that if there is no change in sub-objects, nothing goes here.]
[A new <Sub-object> was added]
-----------------------------------------------
```

```
[An existing  <Sub-object> was deleted]
-----------------------------------------------
[An existing  <Sub-object> was updated]
[List of all modified sub-object fields]
```

1.1.12.2.8.2 Audit Default Description

The default description (the value that TeamConnect puts into the "Text" field of an audit history record) is fairly detailed. In most cases, you do not need to override the default description with your own description.

In the case of Update triggers, the description includes changes for not only the fields of the main object, but also changes for some sub-objects. In addition to the sub-objects specifically named in the table below, every category assigned to an object is also considered a sub-object, for purposes of audit history logging.

Objects and their Sub-objects

Main Object	Sub-objects
Account	(none)
Appointment	Attendee Resource
Contact	Address Email Fax Internet Address Phone Rate Skill Territory
Document	(none)
Expense	(none)
History	(none)
Invoice	(none)
Task	Assignee

Custom Object	Assignee Embedded Custom Object
Involved	(none)
Embedded Custom Object	(none)

1.1.12.2.8.3 Configuring History Object Definition for Audit Tracking

If you want the information captured by audit rules to be in custom fields, you must define these fields under a particular category of the History object. Capturing details in separate fields makes it is easier to export that information using Business Objects® reports.

Note: *If you are creating audit rules for the* ***User*** *or the* ***Group*** *object definition, you cannot define fields for History categories.*

To distinguish audit history records from other history records, create descriptive categories in the History object definition, such as the **Information Change - Lawsuit Key Dates** category, and then select the appropriate category while configuring audit rules. This technique is also useful if you decide to set security by the category rather than preventing user groups from accessing all history records.

The following table shows examples of History object categories and their custom fields that you may configure to be populated by audit rules.

History Object Definition Category and Custom Field Examples

Category name	Custom fields
Information Change - Lawsuit Key Dates	• Old Trial Date • New Trial Date
Information Change - Contact	• Address • Address (old) • Contact Rate • Contact Rate (old) • Email Address • Email Address (old) • Fax Number • Fax Number (old) • Phone Number

	• Phone Number (old)

For details about defining categories, see Using Categories. For details on defining custom fields, see Custom Fields.

1.1.12.2.9 Post Commit Rules

Unlike the other rule types, which are all executed before the user's actions are completed and committed to the database, post commit rules are only executed when it is confirmed that the triggering event has completed successfully. This prevents cases where, for example, a notification email might be sent out even if a transaction failed to commit for any reason.

The qualifier tab and triggers available to post commit rules are the same as those available for custom action rules.

Keep the following points in mind when creating post commit rules:

- Actions in post commit rules are executed outside of a database transaction. No updates to the current record are made within a post commit rule.
- The **Use template** option is not available for post commit rules. The available options are **Use automated Action** and **Notification** (when implemented).

1.1.12.3 Rule Triggers

Triggers are operations that users perform in TeamConnect that affect specific records, such as creating tasks, activating accounts, posting invoices, voiding invoices, deleting contact records, and so on. Each type of rule may place restrictions on certain operations. You select the operation that triggers the rule on the **General** tab of the rule.

1.1.12.3.1 Triggering Events

For a full list of which triggering events are available for each object type and rule type, see the Rule Triggers table.

- **Activate—**The rule is triggered when an account is activated.
- **Assign—**The rule is triggered when a task is assigned.
- **Check In—**The rule is triggered when a document is checked in.
- **Check Out—**The rule is triggered when a document is checked out.
- **Create—**The rule is triggered when an object is created.
    - Rules for the Contact object that use the Create trigger apply when users are creating contacts through a search module. For example, if you make the **Date of Birth** field required, users will be unable to create a contact record through the search module because there is no **Date of Birth** field in the search module.
    - Rules that use the Create trigger for objects that have wizards are executed when the user reaches the end of the wizard and the system attempts to save the new record. If the rule requires that a field be populated, make sure that the user is able to enter a value and knows that the field is required.

- **Deactivate—**The rule is triggered when an account is deactivated.
- **Delete—**The rule is triggered when an object is deleted.
- **Deposit—**The rule is triggered when a budget amount is deposited into an account.
- **Phase Change—**The rule is triggered when a project's phase is changed. When the Phase Change trigger is selected, checkboxes and drop-down lists with further options appear.
    - **From—**The rule is triggered when a project's phase is changed from the selected phase in the drop-down list.
    - **To—**The rule is triggered when a project's phase is changed to the selected phase in the drop-down list.
    - **Transition—**The rule is triggered when the selected phase transition in the drop-down list occurs. This is the default option when the Phase Change trigger is selected.
- **Post—**The rule is triggered when an expense, invoice or task is posted.
- **Transfer From—**The rule is triggered when a budget amount is transferred from an account.
- **Transfer To—**The rule is triggered when a budget amount is transferred to an account.
- **Update—**The rule is triggered when an object is updated.
- **User Invoke—**The rule is triggered when a user with the appropriate rights selects the rule from the More Actions button in an object definition.
- **Void—**The rule is triggered when an expense, invoice or task is voided.
- **Withdraw—**The rule is triggered when a budget amount is withdrawn from an account.

Points To Remember

Keep the following points in mind when creating rule triggers:

- Each rule type may be used with almost all triggers (see the Rule Triggers table), so base your choice of rule type on your business requirements, not the trigger.
- A rule can use more than one trigger. If more than one trigger is associated with a rule, it is possible that the rule will fire more than once for the same event. For example, the **Account** object custom action rule type has an available trigger named **Withdraw**, and it also has the **Update** trigger that is found in all object definitions. If you define a custom action rule that uses both of these triggers, and a Withdraw event occurs, the monetary amounts in the Account record will change, which causes an Update event. So the rule will execute twice, once as a result of the Withdraw trigger and once as a result of the Update trigger.
- Execution order--If an operation applies to multiple rules, the rules types are executed according to type. The following execution order is indicated on the **General** tab of a new rule that has not yet been saved:
    a. Security
    b. Pre-population
    c. Validation
    d. Approval

e. Custom Action

f. Scheduled Action

g. Audit

h. Post Commit

- Each rule of the same type is executed according to the value of their **Order** fields. For example, after all security rules are executed, validation rules are executed, starting with the rule that has the lowest order. Rules of the same type that have the same values in their **Order** fields are executed in alphabetical order.

The following table lists the triggers that are available for each type of rule. A rule can use multiple triggers.

Rule Triggers

Object	Trigger	Security	Pre-population/ Validation	Approval	Custom/ Scheduled Action	Audit	Post Commit
All Objects	**Create** (**Save**, **Save & Close**, Save **& Preview**, or **Save & New** on a new record)		x		x	x	x
	Update (**Save**, **Save & Close**, **Save & Preview**, or **Save & New** on a new record)	x	x		x	x	x
	Delete	x		x	x	x	x
	User Invoked				x		x
Account	**Activate**	x	x	x	x	x	x
	Deactivate	x	x	x	x	x	x
	Deposit	x	x	x	x	x	x
	Transfer From	x	x	x	x	x	x

	Transfer To	x	x	x	x	x	x
	Withdraw	x	x	x	x	x	x
Document	Check In	x	x		x		x
	Check Out	x	x	x	x		x
Expense	Post	x	x	x	x	x	x
	Void	x	x	x	x	x	x
Group	Create		x		x	x	x
	Delete	x			x	x	x
	Update	x	x		x	x	x
	User Invoked				x		x
Invoice	Post	x	x	x	x	x	x
	Void	x	x	x	x	x	x
Project	Phase Change	x	x	x	x	x	x
Task	Assign	x	x	x	x	x	x
	Post	x	x	x	x	x	x
	Void	x	x	x	x	x	x
User	Create		x		x	x	x
	Delete	x			x	x	x
	Update	x	x		x	x	x
	User Invoked				x		x

1.1.12.4 Creating Rules

This topic provides information about rule types, components, and settings to help you create and administer rules.

To create rules through the user interface, determine the following for each rule:

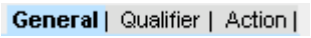
- The object records that the rule affects, such as accounts records.
- The actions the rule takes, such as require approval before a user may perform an operation.
- The conditions that must be met to trigger the rule.
- The specific fields within records that the rule uses.
- The users or groups the rule affects.

Tip: *Consider whether group rights may accomplish your objectives without creating a rule.*

1.1.12.4.1 Defining General Rule Information

General rule information includes several settings, such as the name of the rule, the rule type, and what actions will trigger the rule. This information may be viewed, added, or modified on the **General** tab of the **Rules** screen.

Each rule has its own screen where it is defined. This screen contains the following tabs:



- **General—**General information about the rule, including its name, whether it is active, its rule type, and its trigger(s).
- **Qualifier—**Allows you to define the qualifier that must be met in order for the rule to take effect. For details, see Creating Rule Qualifiers.
- **Action—**For security and validation rules, this tab allows you to enter the message that appears to the user when an operation is denied. For details, see Defining Messages for Security and Validation Rules.

For pre-population rules, the **Action** tab allows you to select the template used to add or populate records. For details, see Using Templates.

For approval rules, the **Action** tab allows you to select the route and specify details about what happens when the route is in effect. For details, see Defining Actions for Approval Rules and Creating Routes.

For scheduled action rules, the **Action** tab allows you to schedule the rule. For details, see Defining Settings for Scheduled Action Rules.

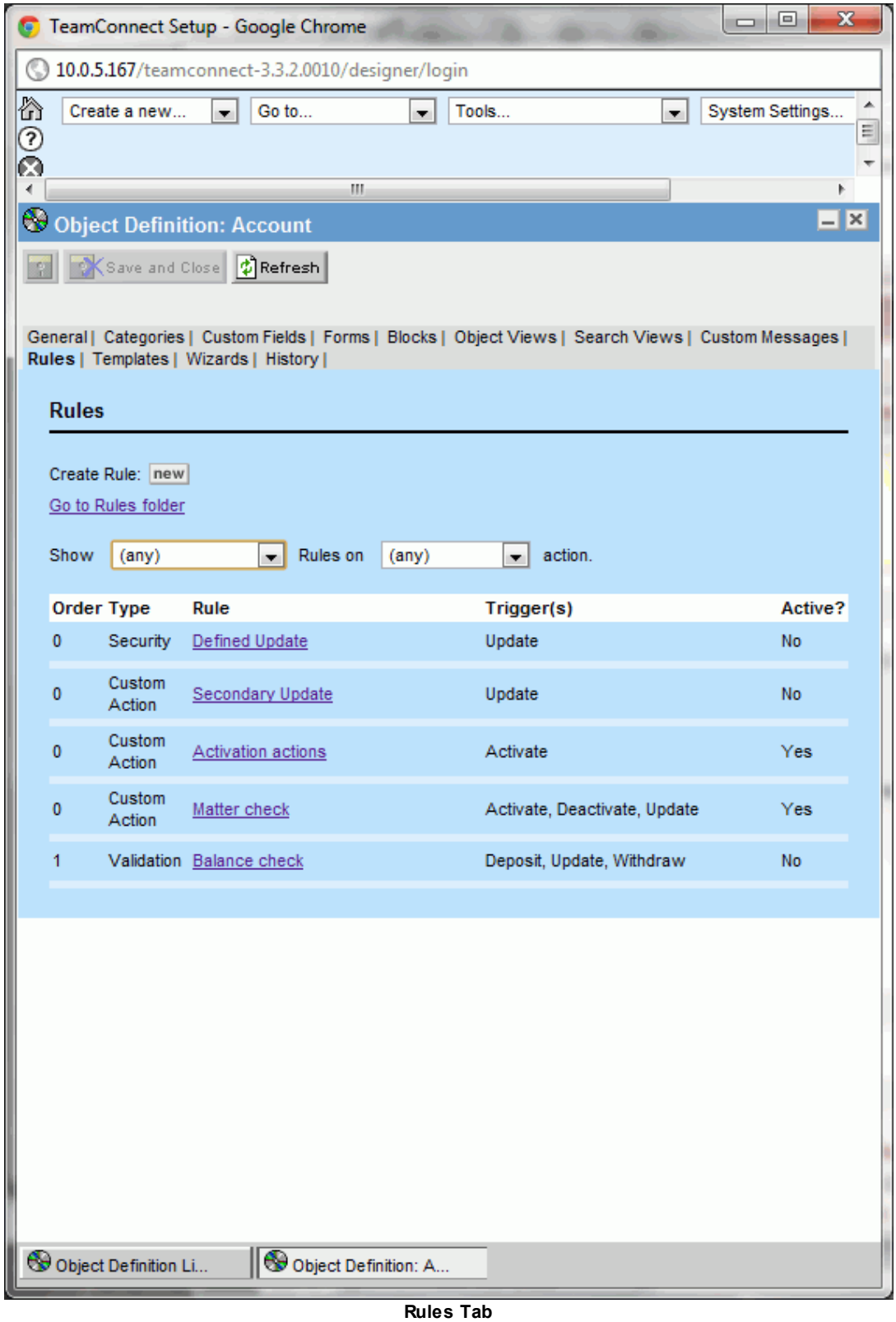
1.1.12.4.2 Accessing the Rules Screen

All rules that are defined within TeamConnect may be created, viewed, and modified on the **Rules** tab of the corresponding object definition.

To access the screen for a new or existing rule

1. Click the **Rules** tab of the object definition for which you want to create or access rules.

   A list of existing rules for the selected object appears, as shown in the following image.


TeamConnect Setup - Google Chrome
10.0.5.167/teamconnect-3.3.2.0010/designer/login
Create a new...
Go to...
Tools...
System Settings...
Object Definition: Account
Save and Close
Refresh
General | Categories | Custom Fields | Forms | Blocks | Object Views | Search Views | Custom Messages | Rules | Templates | Wizards | History |
Rules
Create Rule: new
Go to Rules folder
Show (any) Rules on (any) action.
Order Type Rule Trigger(s) Active?
0 Security Defined Update Update No
0 Custom Action Secondary Update Update No
0 Custom Action Activation actions Activate Yes
0 Custom Action Matter check Activate, Deactivate, Update Yes
1 Validation Balance check Deposit, Update, Withdraw No
Object Definition Li...
Object Definition: A...

Rules Tab

2. If appropriate, sort the list of rules you want to view by either or both of the following criteria:
   - **By rule type—**Select the type from the **Show ___ Rules** drop-down list.
   - **By rule trigger—**Select the operation from the **on __ action** drop-down list.

   The rule list automatically refreshes to show you a list of matching results according to your selections.
3. Click the name of a rule to access it, or click **new** to create a rule.

The following table describes the items on the **Rules** tab.

Rules Tab

Field or column	Description
new	Click to create a new rule.
Go to Rules folder	Click to open the **Rules** folder for the selected object in the Documents area, where all files for custom action and scheduled action rules are stored.
Show __ Rules	Select a rule type by which to filter the list.
on __ action.	Select an action (trigger) by which to filter the list. ***Note:*** *The contents of this drop-down list varies according to the rule type you have selected. For a list of rule triggers, see* *the Rule Triggers table*.
Rule list	Displays all rules that have been created for the selected object that meet your view criteria.
Order	Displays the order in which rules are executed when a user attempts the corresponding action. Rules are executed in order from lowest to highest. The order only pertains to rules of the same type. For example, all security rules are executed first according to order. After all security rules are executed, validation rules are executed according to order, and so on, according to the rule type. For details on execution order, see Points To Remember.
Rule	Displays the name of the rule as a hyperlink, which you may click to view the rule definition.

Trigger(s)	Displays the names of the triggers that are currently associated with the rule. For a list of rule triggers, see the Rule Triggers table.
Active?	Displays whether the rule is currently active in TeamConnect.

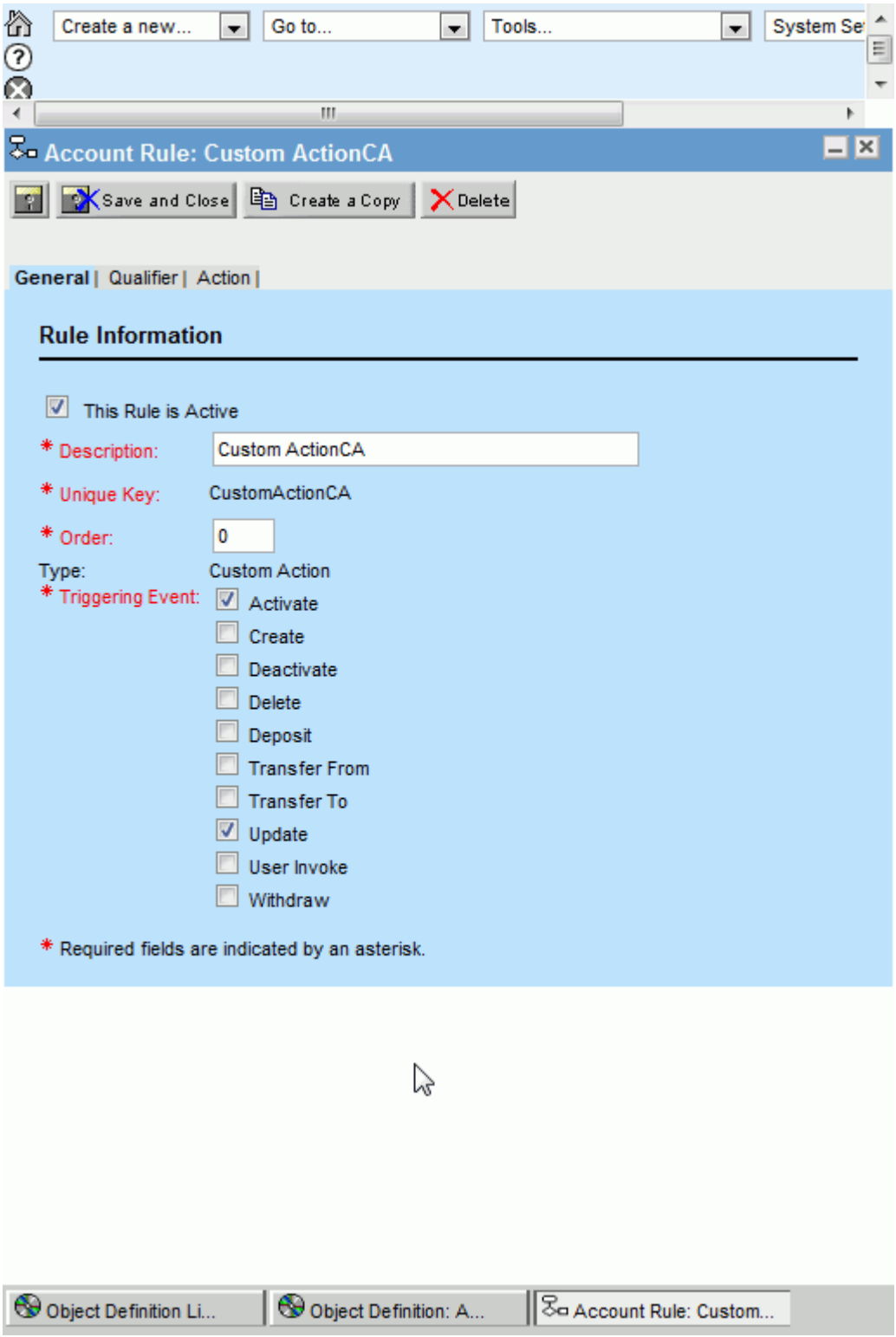
1.1.12.4.2.1 Setting General Rule Information

Before you may define a rule's actions and qualifiers, you must complete the fields on the **General** tab and save the rule.

To set the general rule information

1. Click the **Rules** tab of the appropriate object definition.
2. Click **new** on the **Rules** screen to open a new rule.

   The **General** tab of a new **Rule** screen appears.



General Tab on Rule Screens

3. Enter the necessary information as described in the General Tab on Rule Screen table.

4. Make sure that **This rule is Active** is not selected until you are ready to test the rule.

   Before activating the rule, you must set the necessary information on the **Qualifiers** and **Action** tabs.

5. Click **Save**.

The following table describes the items on the **General** tab of the **Rule** screen.

General Tab on Rule Screen

Field	Description
This Rule is Active	Select to activate the rule after you have finished developing it and you are ready to test it or make it available for users. ***Important:*** *When designing rules, you must always deactivate rules you have not completed or are not using. Having active rules may hinder your ability to design and test other rules.*
Name	Enter a name that clearly and uniquely identifies the rule. This field cannot exceed 250 characters. ***Tip:*** *Avoid similar descriptions for different rules. Emphasize important information, such as the rule trigger, in the description such as the following example:* `When a dispute is closed it cannot be deleted` If this rule is an approval rule, this description appears on the **Workflow** tab of a record that is pending approval for an operation. This helps users to understand why the operation is being sent for approval.
Unique Key	Enter an alphanumeric combination to uniquely identify the rule within the object definition. After you save the rule, the unique code appears as read-only text and you cannot change it. The unique key cannot contain spaces, underscores, commas, punctuation marks, or any special characters and must not exceed 50 characters in length. Although rules within the same object definition cannot have the same unique key, rules belonging to different object definitions may have the same key.
Order	Enter an integer to indicate the order in which the rule must take effect in relation to other rules that are created for the selected object. Organize your rules in the order of their importance in your system, with the most important rule being executed first.

Description	Enter a detailed description of the rule. This field cannot exceed 2000 characters.
Type	Select the radio button next to the type of rule you want to create. For an explanation of each type of rule, see Rule Types. ***Important:*** *After the rule is saved, you cannot change this selection.*
Triggering Event	Select the events, or operations attempted by the user, that you want to trigger the rule. Multiple events can be selected. ***Note:*** *The checkboxes available vary according to the rule type you select. For details, see Rule Triggers.*

1.1.12.4.3 Rule Qualifiers

A *qualifier* for a rule may be defined either in a file that is uploaded to TeamConnect, or directly through the user interface. This section describes qualifiers defined in the user interface. For details about files that are uploaded (custom code), see Writing Automated Qualifiers for Rules.

A qualifier defined through the user interface consists of two parts:

- A list of qualifier items, which identify data values from fields in TeamConnect, that are to be checked by the rule.
- Logic for determining whether the entire qualifier is met: all of the items must be met, any of them must be met, or a more complex logic that you define.

When the rule is triggered, if the qualifier as a whole is met, then the rule executes its action. If the qualifier is not met as you have defined it, the action is not executed.

It is helpful to use this sentence as a model for creating a validation or security rule and defining its qualifier items through the user interface:

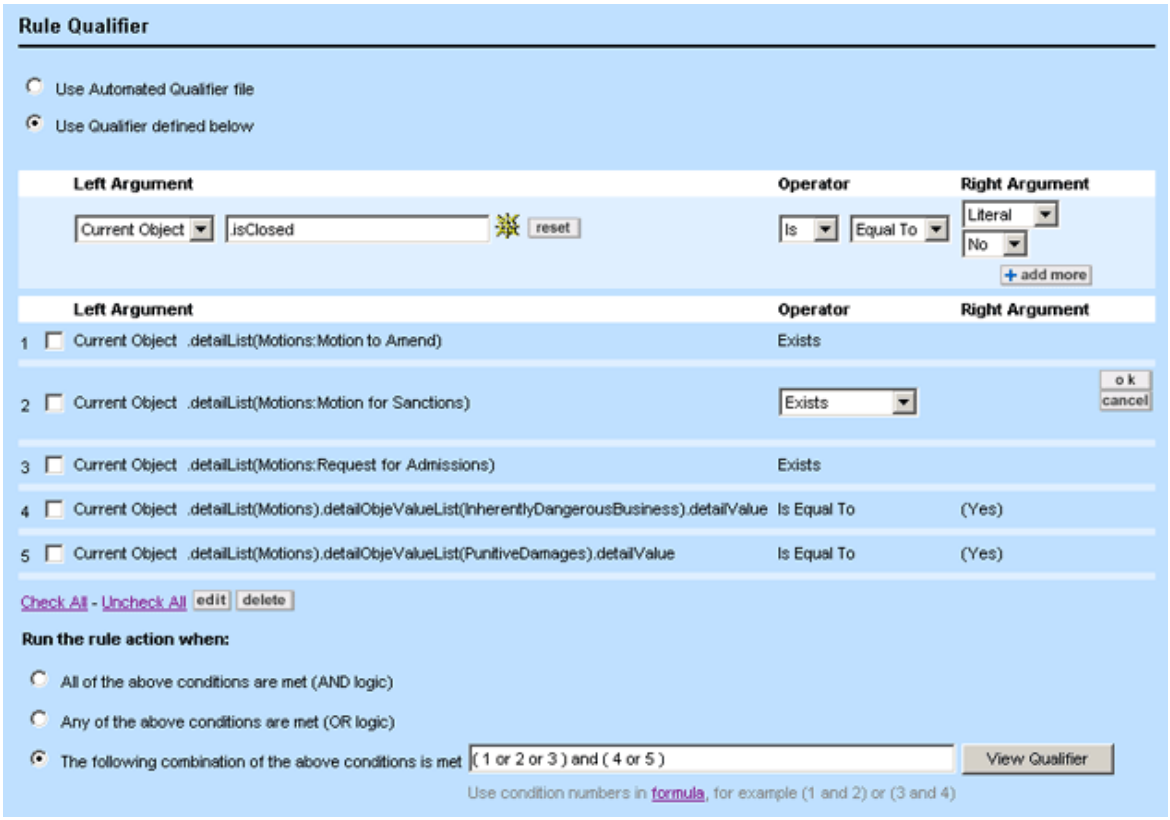
"If the entire qualifier is met, do not allow the user to [update, create, delete] the record."

Following the construction of this sentence when creating qualifiers will help you write them accurately.

Important: *If an active rule has no qualifier information defined, the rule action is always executed when the rule's triggering events occur, whether by a user, wizard, or another rule.*

1.1.12.4.3.1 Qualifier Tab Information

You may view, create, modify, or delete the qualifier information for a rule on the **Qualifier** tab of the respective **Rules** screen of an object definition ().



Qualifier Tab on Rule Screens

The following table describes the items in the **Qualifier** tab in **Rule** screens.

Qualifier Tab on Rule Screens

Field	Description
Use Automated Qualifier file	Select this option to use a Java class or JavaScript file as the rule's qualifier. When this option is selected, the screen displays a drop-down list labeled **Use this Automated Qualifier**, where you may select the appropriate Java or JavaScript rule qualifier. For details about automated qualifiers, see Writing Automated Qualifiers for Rules.
Use Qualifier defined below	Select this option if you want to use the user interface to define the qualifier conditions and logic.
Left Argument	Allows you to create a path to the attribute that you want to use in the qualifier condition.

	Select whether you want to create a path to an attribute that is associated with the **Current Object** or the **Current User**: • Select **Current Object** if you want to use a field in TeamConnect as a qualifier (for example, the Current Phase of the record that the user is attempting to update). • Select **Current User** if you want the rule to check the identity of the current user working with a record (for example, to verify that the user who is attempting the operation is a member of a specified group). Click the **Navigator** icon to open Object Navigator, the tool for creating paths in TeamConnect, and edit the path displayed in the **Attribute** field, if appropriate. To create a path, you must understand the object model as described in Object Model: Read This First and in the additional tables it points to.
reset	Clears the path that appears in the Object Navigator field. This button appears after an option has been selected from the drop-down list in the Left Argument.
Operator	Select the appropriate operator settings for the qualifier condition. For details, see Available Operators in Qualifier Items.
Right Argument	Select the attribute, literal value, or other setting that you want to compare to the attribute that you selected for the left argument. For details, see Right Argument Options in Qualifier Items.
Qualifier Item List	Displays a list of qualifier items that have been added to the rule. These are the conditions used by the rule, in combination with the logic that you specify, to determine whether to prevent the operation that is being attempted, or whether the rule's action must be run. Select or clear the corresponding check-boxes to **edit** or **delete** qualifier items.
Run the rule action when	Select how you want the defined qualifier items to be evaluated by the rule. For details, see Qualifier Logic.

1.1.12.4.3.2 Qualifier Items

To define a qualifier item in Designer, you specify all of the following:

- The left argument
- The operator
- The right argument

These three pieces together form a statement, such as these:

Sample Qualifier Statements

Left argument	Operator	Right argument
The value selected in the Loss Type list field	is	Fire Damage
The current user	is not	a member of the Outside Counsel group
The Trial Date	is changed	from any date to any date
The Expiration Date field	is populated	[none]

If you add these statements as qualifier items, or conditions, the rule will evaluate the statements to determine whether the rule's action must be executed when a user triggers the rule.

The type of data being compared in the left and right arguments must match. For example, if your left argument identifies a list field, then your operator and right argument options allow you to compare the selection with a selection made in another list field or to a literal list value.

The **Designer** user interface is designed to deny illogical combinations of left arguments, operators, and right arguments, such as: "The Trial Date is the Outside Counsel group."

Tip: *If you find a rule executing in a situation where you did not expect it to, carefully check how you have constructed your qualifiers. In some cases, when the left qualifier evaluates to an unknown value and the right qualifier also evaluates to an unknown value, the rule engine will consider this a match. For example, consider an Audit rule triggered by Create that affects Involved records, and has a qualifier with these characteristics:*

```
Left qualifier = inactiveDate
Operation = X Days from Today
Right qualifer = (no value)
```

In this case, when you create a new Involved record that is active, the rule will execute. That's because the value of inactiveDate is null, since the record has never been inactivated. And because the right qualifier was specified incorrectly (an integer should have been entered, but wasn't), the right qualifier's value is also null. Thus the rule engine considers the two nulls a match, and the rule executes. You should fix the incorrect specification of the right qualifier. Also, to guard against such inadvertent errors, you could add a second qualifier, testing the isPopulated condition for the inactiveDate field, so that the rule would only execute when that field was actually populated.

For details about the options available when constructing qualifier items for each type of data, see:

- Available Operators in Qualifier Items
- Right Argument Options in Qualifier Items

The operator in a rule qualifier statement determines how the left and right arguments are compared. It may have several components, depending on the type of data value you are working with in the qualifier (text, lookup table value, date, and so on).

The following table describes TeamConnect operators and the corresponding types of data that may be compared.

Rule Qualifier Operator Options

Selection	Description	Data types where operator selection is available					
		Text	Number	Date	List	Check-box	Related object
Is	Determines whether the qualifier statement is a positive or negative statement.	x	x	x	x	x	x
Not		x	x	x	x	x	x
Equal To	Evaluates whether the left argument and right argument are the same value.	x	x	x	x	x	x
Populated	Evaluates whether the field identified in the left argument has a value. Do not use this selection to check whether a field that was previously null now has a value. Instead, use Changed from No Value.	x	x	x	x		x
Changed	Evaluates whether the value for the field identified in the left argument has changed. It compares the old value in the database to the value in the record when rule is triggered. ***Tip:*** *This option is useful for triggering audit rules.*	x	x	x	x	x	x
Begins With	Evaluates whether the text value in the field identified in the left argument begins with, ends with, or contains	x					

Ends With	the text that is identified in the right argument.	x					
Contains		x					
Less Than	Evaluates the number value identified in the left argument compared to the number value in the right argument. For example, you may specify that if a dollar amount **Is** - **Greater Than Or Equal To** 5000, send the action for approval.		x				
Less Than Or Equal To			x				
Greater Than Or Equal To			x				
Greater Than			x				
Exists	These options appear when you select a particular category by creating a path such as `.detailList.(Motions)`. Evaluates whether the category has been added to the record.						
Does Not Exist							
Item Added	Evaluates whether a related object (such as involved) or sub-object (such as assignee) was added, deleted, or modified.				x		
Item Deleted					x		
Item Modified					x		
After	Compares the date in the left argument to the date in the right argument. An additional field appears when you select one of the options that requires you to specify a number of days. Enter the value for **X** in this			x			
Before				x			
After The Next X Days				x			

Within The Next X Days				x			
Within The Last X Days				x			
Before The Last X Days				x			

Depending on the data type of the qualifier item that you are adding to your rule qualifier, the selections that are possible for the right argument are automatically provided to you.

The following table describes the possible selections for the right argument of a rule qualifier.

Rule Qualifier Right Argument Options

Selection	Description	Data types for which selection is available					
		Text	Number	Date	List	Check-box	Custom Object/ Involved
Literal	Select this option when you want to specify a value to compare to the field identified by your left argument. For example, select **Literal** for a qualifier item that checks whether the invoice total is greater than 5000. You must also select **Literal** when you want to check whether a certain item is selected in a List field.	x	x		x	x	x
Attribute	Select this option when you need to compare the values of two fields in TeamConnect. After you select **Attribute**, you must first select	x	x	x	x	x	x

	Current Object or **Current User**, and then use Object Navigator to identify the field to which you want to compare the field identified by your left argument. For example, if you want to compare the Contact of a contact-centric project to the Contact identified in a custom field of type Involved, select **Attribute** to identify the custom field.						
Current Object	Select one of these options after you have selected **Attribute** and you need to identify a field in TeamConnect to which you are comparing the field in your left argument. For details about using Object Navigator, see Using Object Navigator.	**x**	**x**	**x**	**x**	**x**	**x**
Current User		**x**	**x**	**x**	**x**	**x**	**x**
Any Value	These options work in conjunction with the **Changed** option to form a statement, such as **Is changed from any value to this value (California)**.	**x**	**x**	**x**	**x**	**x**	**x**
No Value		**x**	**x**	**x**	**x**		**x**
This Value		**x**	**x**		**x**		**x**
Today	The date that the rule is being triggered.			**x**			
This Date	Allows you to compare the date in the left argument to a date that is identified in another field.			**x**			
From Now	Allows you to specify a number of days from the date that the rule is being triggered.			**x**			

From This Date	Allows you to specify a number of days from a date that is identified in another field.			**x**			
Yes	These options correspond to a check-box being selected or not selected.					**x**	
No						**x**	

1.1.12.4.3.3 Qualifier Logic

The **Designer** user interface is designed for you to complete the qualifier logic in stages. After listing one or more qualifier items on the **Qualifier** tab, the following options appear, allowing you to specify when the rule action runs:

- All of the conditions are met.
- Any of the conditions are met.
- The conditions are met according to the logic that you specify.

The "logic" option allows you to specify a formula for how the rule evaluates the conditions. You refer to a qualifier condition using the number that is automatically generated on the batch screen for each item (see). For example, once you create a list of five qualifier conditions, you might create the following formula:

```
(1 or 2 or 3) and (4 or 5)
```

Points To Remember

- Keep the following points in mind while working with qualifier logic:
- Define all of the qualifier conditions that you need in your rule qualifier before writing the formula.
- The only words that you may use in your formula are "and" and "or."
- Use parentheses () to logically group conditions. You may also nest parentheses when needed. Make sure to use complete pairs of parentheses.
- If you do not group the conditions where grouping is necessary, grouping is automatically applied by TeamConnect, and you will have to verify that the appropriate logic is achieved. For example:

  ```
  1 or 2 and 3
  ```

  is automatically converted to:

  ```
  1 or (2 and 3)
  ```

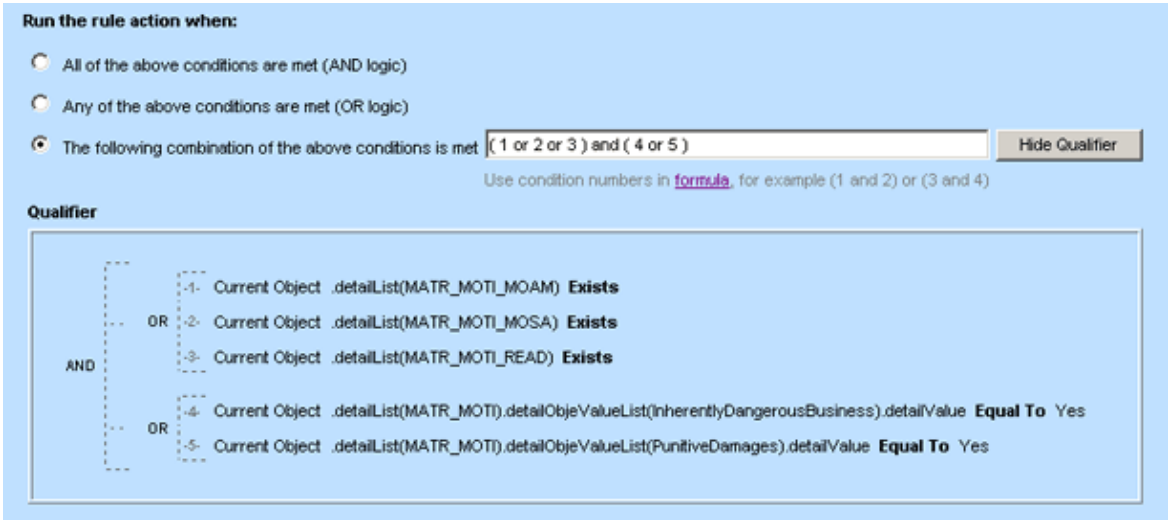
  It is best to apply the grouping yourself to ensure that it is correct.
- If necessary, your formula may repeat the same qualifier condition. For example, the following formula is valid:

  ```
  (1 and 2 and 3) or (1 and 2 and 4)
  ```

Viewing Completed Qualifier

Once you have added all of the necessary qualifier conditions and have written the formula that references those conditions, you may view the complete qualifier expression by clicking **View Qualifier**.

If you modify your qualifier conditions or formula, you may refresh the display of your qualifier by clicking **Save**, or by clicking **View Qualifier** again. The following figure shows an example of qualifier logic:



Qualifier Logic Options

The following table describes the items in the qualifier logic section.

Qualifier Logic Options

Field	Description
Run the rule action when	Specify when the rule action must be run based on the qualifiers. You may specify AND or OR logic, or a formula using the qualifiers. See Qualifier Logic.
View Qualifier/Hide Qualifier	Click to view or hide the qualifier logic display.
formula	Click this hyperlink to view additional help on how to write the formula for a rule qualifier.
Qualifier	Displays a visual representation of your entire rule qualifier according to the formula that you have provided. You cannot edit the qualifier in this area. It is provided to help you determine whether the formula you have written is correct.

1.1.12.4.3.4 Creating Rule Qualifiers

Creating rule qualifiers requires an understand of the following items:

- **Rule components—**Creating appropriate qualifiers is only possible when you understand how rules function, including all of the following:
  - Rule Types
  - Rule Triggers
  - Qualifier Items
  - Qualifier Logic
- **TeamConnect object model—**The object model is described in detail in Object Model: Read This First and the additional reference tables it points to.
- **Creating paths—**You need to understand how to create paths to attributes using Object Navigator. For details, see Using Object Navigator.

To create a qualifier for a rule

1. Plan your rule so that you know which attributes and conditions you need to identify in TeamConnect.
2. Open the **Qualifier** tab of the appropriate rule.

   Important: *Before you may create the qualifier, you must save the rule with the correct information on the* ***General*** *tab.*

3. Create the qualifier conditions as described in Qualifier Items. You may refer to Qualifier Item Examples for examples that may be used as models for your rules.
4. Select whether you want to run the rule when:
   - **All of the above qualifiers are met (AND logic).** (This is the default.)
   - **Any of the above qualifiers are met (OR logic).**
   - **The following combination of the above conditions is met.**
5. If you selected one of the first two options, this action completes your rule's qualifier. Click **Save**.
6. If you selected the third option, use the text field that appears to write a formula that uses the qualifier conditions that you have defined. For details, see Qualifier Logic.
7. Click **View Qualifier**.

   Your entire qualifier expression appears, showing the logical grouping that you have defined in your formula. For an example, see Viewing Completed Qualifier.
8. Click **Save**.

   To complete a rule, you must define its action. For details about the rule action for various rule types, see Rule Actions.

1.1.12.4.3.5 Qualifier Item Examples

The following are examples of qualifier items for rules. Use these examples as models for constructing your own rule qualifiers.

Each example demonstrates exactly what you would click, select, and type to define the qualifier item, using numbered steps that start with the left argument selections and finish with the right argument selections.

These examples use hypothetical values for categories, custom fields, user group names, and so on.

Rule Qualifier Item Examples

Can be used for:	What this qualifier item is checking	Left argument selections	Operator selections	Right argument selections
Any object	Checks whether the current user who is attempting the operation is a member of the **Paralegals** group.	1 Current User 2 Click the Object Navigator icon 3 groupMemberList-> (traverse) 4 group-> (ok)	5 Is 6 Equal To	7 Literal 8 Paralegals
Any object	Checks whether the current user is Bob Johnson.	1 Current User	2 Is 3 Equal To	4 Literal 5 Bob Johnson
Any object	Checks whether the current user is the same user who created the record.	1 Current User	2 Is 3 Equal To	4 Attribute 5 Current Object 6 Click the Object Navigator icon 7 createdBy-> (ok)
Contact-centric custom object	Checks whether the contact selected in the contact-centric field of the project (such as the **Insured**) is the contact selected in a custom field of type Involved (such as **Requestor**).	1 Current Object 2 Click the Object Navigator icon 3 contact-> (ok)	4 Is 5 Equal To	6 Attribute 7 Current Object 8 Click the Object Navigator icon 9 detailList-> (traverse) 10 RequestedBy-> (traverse) 11 contact-> (ok)

Any object	Checks whether the value of a custom field **Trial Date** has changed (for example, to trigger an audit rule).	1 Current Object 2 Click the Object Navigator icon 3 detailList-> (traverse) 4 Select the category (traverse) 5 TrialDate (ok)	6 Is 7 Changed	8 Any Value 9 Any Value
Custom object (such as a matter)	Checks whether an assignee with the role **Outside Counsel** has been added to the record.	1 Current Object 2 Click the Object Navigator icon 3 assigneeList-> (traverse) 4 Outside Counsel (ok)	5 Not 6 Item Added	None
Custom object (such as a matter)	Checks whether the phase is changed from some other phase to **Closed-Pending**.	1 Current Object 2 Click the Object Navigator icon 3 currentPhaseType -> (ok)	4 Is 5 Changed	6 From: Any Value 7 To: This Value 8 Closed- Pending
Any object	Checks whether the custom field **Disposition Date** has no value (is null).	1 Current Object 2 Click the Object Navigator icon 3 detailList-> (traverse) 4 Select the category (traverse) 5 Disposition Date (ok)	6 Not 7 Populated	None
Any object	Checks whether the **Suit Served** date is after the **Suit Filed** date (both custom fields).	1 Current Object 2 Click the Object Navigator icon	6 Is 7 After	8 This Date 9 Current Object

		3 detailList-> (traverse) 4 Select the category (traverse) 5 Suit Served (ok)		10 Click the Object Navigator icon 11 detailList-> (traverse) 12 Select the category (traverse) 13 Suit Filed (ok)
Any object	Checks whether the **Incident Date** was during the last 30 days. Incident Date is a custom field.	1 Current Object 2 Click the Object Navigator icon 3 detailList-> (traverse) 4 Select the category (traverse) 5 Incident Date (ok)	6 Is 7 Within The Last X Days 8 30	9 From Now
Any object	Checks whether the **Loss Estimate** is greater than or equal to 1000 dollars. Loss Estimate is a custom field.	1 Current Object 2 Click the Object Navigator icon 3 detailList-> (traverse) 4 Select the category (traverse) 5 Loss Estimate (ok)	6 Is 7 Greater Than Or Equal To	8 Literal 9 1000
Task or Expense	Checks whether the project associated with the record is in **Closed** status. (This is the record status—not the phase.) ***Tip:*** *You may use this qualifier with the Post rule trigger to*	1 Current Object 2 Click the Object Navigator icon 3 project-> (traverse) 4 isClosed (ok)	5 Is 6 Equal To	7 Literal 8 Yes

	prevent posting to a closed project.			
Invoice	Checks the type of invoice, since some types may be subject to more stringent verification and approval than other types are. You may want to restrict your rule to a specific invoice type, or you may want to exclude a specific type (for example, "Not Shadow".)	1 Current Object 2 Click the Object Navigator icon 3 typeIID (ok))	4 Is 5 Not	6 Standard 7 Accrual 8 Credit Note 9 Shadow
Task or Expense	Checks whether the Current Phase of the project associated with the record is **Pending**. ***Tip:*** *You may use this qualifier with the Post rule trigger to prevent posting to a project when it is in an inappropriate phase for posting.*	1 Current Object 2 Click the Object Navigator icon 3 project-> (traverse) 4 currentPhaseType -> (ok)	5 Is 6 Equal To	7 Literal 8 Pending
Contact	Checks whether there are any phone numbers added to the contact.	1 Current Object 2 Click the Object Navigator icon 3 phoneList-> (ok)	4 Not 5 Populated	None

1.1.12.4.3.6 Limitations for User Interface Qualifiers

If your rule requirements call for qualifiers that cannot be created using Designer, you must write automated qualifiers in Java or JavaScript. For details about creating automated qualifiers, see Writing Automated Qualifiers for Rules.

For example, the following qualifiers cannot be created using Designer and therefore must be written as automated qualifiers:

- Check the values for more than one attribute of a single sub-object.

  For example:

  - A qualifier that checks whether there is an active assignee of the type Underwriter.
  - A qualifier that checks whether an invoice line item has a certain task category and the amount of the same line item.
  - A qualifier that checks the state and the city of the same address of a particular type from a contact's list of addresses. This is only possible if you are checking the state and city of a contact's default address.

- Check the values for more than one attribute of a single related object from a list of related objects.

  For example:

  - A qualifier that checks whether there is an involved (through involvedList->) with a certain role, and in the same involved's contact record, whether it has a certain firm as its company in the Company field.
  - A qualifier that checks whether there is a task related to the project with a certain category and that same task is not completed.

- Check the value in a record that is not somehow related to the current object for which the rule is written. This requires searching, which is possible in rule qualifiers only through custom code.
- Perform complex calculations.

1.1.12.4.4 Rule Actions

A rule action is what a rule performs when the entire qualifier is met. The fields on the **Action** tab depend on the rule type:

- The action for security and validation rules is to deny the operation that is attempted by the user if the qualifier conditions in the rule are met. If an operation is denied by a security or validation rule, a message appears.

  You may view, create, or modify a rule's rejection message on the **Action** tab. For details, see Defining Messages for Security and Validation Rules.

- The action for approval rules is to follow a Route of approvers who decide whether to deny or allow the operation.

  You must set various parameters for an approval rule, including which route to use, on the **Action** tab of the rule. For details, see Defining Actions for Approval Rules.

- The action for custom action or scheduled action rules is defined using Java or JavaScript. You select the file that defines the action on the **Action** tab of the rule.

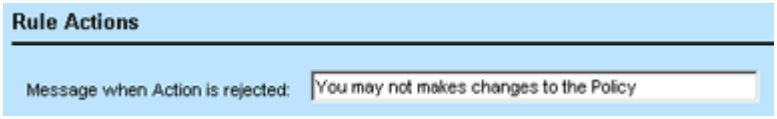
  You may also need to make additional selections on the **Action** tab, depending on the rule type and whether additional fields appear that are specific to the custom rule. For details about the fields available for scheduled action rules, see Defining Settings for Scheduled Action Rules.

1.1.12.4.4.1 Defining Messages for Security and Validation Rules

When the qualifier for a security or validation rule is met, the operation attempted by the user is automatically rejected. You are only required to enter a message that appears for users when the operation is rejected. If the operation is approved, the rule does not display a message to the user.

Keep the following points in mind when writing messages for rules:

- The message may not only help the user understand what to do, but it may also help you identify which rule was triggered when users are having difficulties. For example, you may include a reference number that identifies the rule.
- If you do not specify a message, a generic message box is shown to users. Make sure to include a message that is descriptive and includes a possible solution for the user.
- Messages are limited to 2000 characters. If you type more characters than that, the extra characters will be truncated without warning.



Action Tab on Security or Validation Rule Screens

To specify a denial message for a security or validation rule

1. Open the **Action** tab of the rule whose message you want to specify.
2. Enter a message in the **Message when Action** is rejected field.

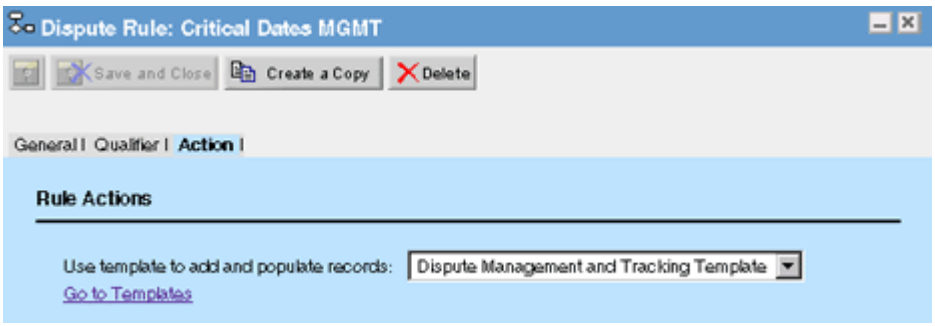
   For example:

   ```
   You must include an Attorney and a Main Assignee to save this Litigation
   record.
   ```
3. Click **Save** to save the message as part of the rule.

1.1.12.4.4.2 Defining Actions for Pre-population Rules

When a pre-population rule is triggered, the rule uses its template to prepopulate specific fields or child records. For more information about creating templates for an object definition, see Using Templates.

To be able to specify a template, you must create at least one template defined for use with rules. If you do not specify a template, no fields are prepopulated.



Action Tab on a Pre-population Rule

To specify the template used by a pre-population rule

1. Open the **Action** tab of the pre-population rule whose template you want to specify.
2. From the **Use template to add and populate records** drop-down list, select the appropriate template.

   If no list appears, no templates have been defined to be used with rules for the object definition.
3. Click **Save** to associate the template with the rule.

The following image shows the **Action** tab when no templates are available for the object definition. **No Item Selected** appears as read-only text and the **Go to Templates** link allows you to open the **Templates** tab of the object definition.

Action Tab of Rule Screen with No Available Templates

For more information about templates, see Using Templates.

1.1.12.4.4.3 Defining Actions for Approval Rules

When defining the action of an approval rule, Use the Action tab to perform the following actions:

- Select the route that the approval rule follows when the qualifier conditions are met.
- Specify a Process Manager. The user group that you specify as Process Manager must contain at least one active user.
- Specify settings, either at the route level or stop level, for the rule as follows:
  - Whether or not to allow the record to be sent for review.
  - The expiration action at the route level or for each individual stop, such as automatically approve, automatically reject, or send to Process Manager.
  - How to handle approvers who appear at multiple stops.
  - Who may update the record while it is pending approval.

The settings available for the approval rule action depend on whether you want to set route-level parameters or stop-level parameters.

About Process Managers

Note: *Although this documentation discusses Process Manager as if it were a single person, a Process Manager is always defined via a user group. You cannot assign an individual user to the Process Manager role. All members of the assigned group receive email related to*

workflow requests, and all have equal authority to perform Process Manager tasks. A task requires only one group member to complete it; whoever handles it first handles it for the entire group.

An approval rule always requires a Process Manager. The default value for Process Manager is the group "Workflow Process Manager." You may wish to change the default to some other user group when editing the rule. The Process Manager becomes responsible for monitoring approval processes triggered by that rule on the **My Workflows** screen. The Process Manager may perform the following actions:

- Reject the requested operation by returning it to the user who triggered it.
- Restart the approval process at the first stop of the route.
- Reassign any current approval requests to any TeamConnect user.
- Reassign the approval request to approvers in the current stop who rejected the requested operation.

About Route-level versus Stop-level Parameters

The main difference between Stop-level versus Route-level parameters is that for Stop-level parameters, you may set specific options, such as allowing approvers to create an impromptu reviewer list, allowing stop members to update records (pending approval), and setting the approval time limit and expiration action on a per stop basis. If you choose Route-level parameters, those options are available, but applied universally to all stops for that route.

To define the action settings for an approval rule

1. Open the **Action** tab of the approval rule.
2. Select the route of approvers that you want this rule to follow from the drop-down list. For details about routes, see Creating Routes.
3. Type or select the rest of the settings as described in the Action Tab on Approval Rule Screens tables for Stop-level Parameters Selected or Route-level Parameters Selected.
4. Click **Save** to save the settings as part of the approval rule.

For details about how approval rules function, see Approval Rules. For more details about routes, see Creating Routes.

The following table describes the fields in the **Action** tab of approval rule screens when using stop-level parameters.

Action Tab on Approval Rule Screens (Stop-level parameters selected)

Field	Description

Approval route	Select an existing route from the drop-down list to specify the users who must authorize the request. If this area displays (No Items to Select), then you must first create a route. For details, see Creating Routes.
Process Manager	Select a user group from the drop-down list. (You may first need to define a new group and/or add users to a group, as explained in "Account Administration" chapter of the TeamConnect Administration Guide.) The group must contain at least one active user. When you add a Process Manager, any user in that group may reassign processes, restart processes, or return requests to requestors for the following statuses: **Action on reject - Send to Process Manager** (see below in this table) **Expiration action - Send to Process Manager** (see below in this table)
Action on reject	Select one of the following: • **Return to requestor—**This stops the approval process. The requestor will still have the option to resubmit the approval request. • **Send to first stop—**If an approver at the first stop rejects the operation, the approval process will end. If an approver at the second stop or later rejects the operation, the approval will be pending for the first stop's approver. No matter where in the route the rejection took place, the request is rejected. The Workflow block of the record indicates who rejected the operation and the date and time. • **Send to Process Manager—**Select to allow a Process Manager to reassign the request to a new user (non-approver), restart the approval process, or return the request to the requestor.
Action on return to requestor	This field is used when the previous field, Action on Reject, has the option "Return to Requestor" selected. If Action on Reject has a different option, the setting in this field is ignored. Select one of the following: • **No Action—**The approval is marked as rejected and it is returned to the requestor with no other action taken. • **Use template—**The approval is marked as rejected and the template that is selected in the drop-down list is executed. The link **Go to Templates** allows you to view and manage templates, then return to this page and choose a template to be executed with this action. • **Use automated action—**The approval is marked as rejected and the automated action that is selected in the drop-down list is

	executed. The link **Go to Automated Action Folder** allows you to view and manage automated actions, then return to this page and choose an automated action to be executed with this rule action.
If the same Approver is at multiple stops	Select one of the following: • **Approver approves at each assigned stop—**The assigned approver must approve at all assigned stops. This is the default option. • **Approver approves once and is skipped in subsequent stops—**Once the approver approves at the first stop, all subsequent stops where this user is an approver are marked "Approved" with comments populated. Workflow functions as if the user manually approved each of these stops. ***Notes:*** *This option only applies to actual approver action and does not apply if stop members are auto-approved via an expiration date.* Also, if a stop member approves under this option, and later, a process manager assigns the same user to a stop, the stop member must manually approve the new assignment and approval automation will not overwrite the previously-automated approval.
Use Stop-level parameters	If you select the Use Stop-level parameters option, you may define workflow approval behavior per stop.
Options available prior to approval	You may select one or more of the following for each route stop: • **Send for review—**(optional) Select this option to allow the approver the option to assign reviewers to the operation before he or she approves. • **Send for additional approval—**Select this option to allow the approver, before approving, to send the request to another user. If the new approver approves the request, the request continues to the original approver. If the new approver rejects the request, the specified **Action on Reject** occurs. • **Update records pending approval—**If approval is pending for one or more approvers at a stop level, then any of the approvers at that level have edit rights for the tabs and fields in the record. This field is mutually exclusive with the "**All users that are approvers can update"** option present in the **Options available upon approval** section. • **All users that are approvers can update—**All users on the route will have edit rights for the tabs and fields in the record. This field is mutually exclusive with the "**Update records pending approval"** option present in the **Options available**

	prior to approval section. ***Note:*** *This selection is only available if all users are required on the stop.*
Options available upon approval	You may select the following for each route stop: • **Send for additional approval—**Select this option to allow the approver, after approving, to send the request to another user. If the new approver approves the request, the request continues to the next stop. If the new approver rejects the request, the specified **Action on Reject** occurs. If the rule uses a route with conditions, this settings does not appear on your rule.
Approval time limit	You may set the number of days approvers in each stop have to respond. This number is converted into exact hours. For example, entering 5 gives the approvers five days, or 120 hours, to make a decision concerning the action. So if the rule is invoked on October 15, 2008 at 3:00 pm, the approvers have until October 20, 2008 3:00 pm to approve the action. The countdown starts when the operation is first attempted by the user. You must use a value other than zero when specifying the length of an approval process. Otherwise, the approval period expires almost immediately.
Expiration action	You may select one of the following for each route stop: • **Send to Process Manager—**Select this radio button to allow the Process Manager to reassign or reject a request after the approval time limit has passed. • **Auto-approve—**Select this radio button to allow the operation to be performed after the approval time limit has passed. • **Auto-reject—**Select this radio button to prevent the operation from being performed after the approval time limit has passed.
Email Actions	The **Allow approval/rejection via email** field is a check-box you select if you want users who receive approval requests during the stop to receive an email from which they can approve or reject the request. This setting also allows them to enter internal comments and comments to the requestor after approving or rejecting through email.
Email Template	The drop-down contains all email templates you can select as the approval request email users receive during this stop. If you do not

	select a template, TeamConnect uses a default email template. You can design an approval request email from the **Notifications** page of **Admin Settings**. See Notification Settings for more information.
Hold Conditions	Hold conditions allow you to ensure approvals on a certain stop are contingent on meetin predetermined stipulations. Use the Hold Condition message entry field to provide users with detail on why the approval is blocked. For more details, see Hold Conditions. ***Note:*** *Feature only available in TeamConnect 4.0.8+.*

The following table describes the fields in approval rule screens when using route-level parameters.

Action Tab on Approval Rule Screens (Route-level parameters selected)

Field	Description
Approval route	Select an existing route from the drop-down list to specify the users who must authorize the request. If this area displays (No Items to Select), then you must first create a route. For details, see Creating Routes.
Process Manager	Select a user group from the drop-down list. When you add a Process Manager, any user in that group may reassign processes, restart processes, or return requests to requestors for the following statuses: **Action on reject - Send to Process Manager** (see below in this table) **Expiration action - Send to Process Manager** (see below in this table)
Action on reject	Select one of the following: • **Return to requestor**--Stops the approval process. The requestor will still have the option to resubmit the approval request. • **Send to first stop—**If an approver at the first stop rejects the operation, the approval process ends. If an approver at the second stop or later rejects the operation, the approval will be pending for the first stop's approver(s). • **Send to Process Manager—**Select to allow a Process Manager to reassign the request to a new user (non-approver), restart the approval process, or return the request to the requestor.

If the same Approver is at multiple stops	Select one of the following: • **Approver approves at each assigned stop—**The assigned approver must approve at all assigned stops. This is the default option. • **Approver approves once and is skipped in subsequent stops—**Once the approver approves at the first stop, all future stops where this user is an approver are marked "Approved" with comments populated. Workflow will function as if the user has manually approved each of these stops. ***Notes:*** *This option only applies to actual approver action and does not apply if stop members are auto-approved via an expiration date*. Also, if a stop member approves under this option, and later, a process manager assigns the same user to a stop, the stop member must manually approve the new assignment and approval automation will not overwrite the previously-automated approval.
Use Route-level parameters	This option applies a general set of workflow approval settings across all stops for the selected route. See more details below.
Options available prior to approval	You may select the following for the route: • **Send for review—**(Optional) If you select **Use Route-level parameters**, you may select this option to allow the approver the option to assign reviewers to the operation before the approver approves. • **Send for additional approval—**Select this option to allow the approver, before approving, to send the request to another user. If the new approver approves the request, the request continues to the original approver. If the new approver rejects the request, the specified **Action on Reject** occurs.
Options available upon approval	You may select the following for the route: • **Send for additional approval—**Select this option to allow the approver, after approving, to send the request to another user. If the new approver approves the request, the request continues to the next stop. If the new approver rejects the request, the specified **Action on Reject** occurs. If the rule uses a route with conditions, this settings does not appear on your rule.

Updates to records pending approval	If you select Use Route-level parameters, you may select one of the following: • **Not allowed**. While the request is pending approval, no contacts have edit rights for the tabs and fields related to that object. • **Allowed to approvers of current stop**. The approval request is sent to all members of the group. While the request is pending approval, any approver from the current stop has update rights for the tabs and fields in the record. ***Note:*** *If the Route Stop has been set to require all members to approve, and this option is selected, the stop members will not be able to update the record.* • **Allowed to anyone**. While the request is pending approval, any approvers have update rights for the tabs and fields in the record.
Approval time limit	(Optional) Enter the number of days approvers have to respond. This number is converted into exact hours. For example, entering 5 gives the approvers five days, or 120 hours, to make a decision concerning the action. So if the rule is invoked on October 15, 2001 at 3:00 pm, the approvers have until October 20, 2001 3:00 pm to approve the action. The countdown starts when the operation is first attempted by the user. It does not reset each time a route stop is completed.
Expiration action	You may select one of the following: • **Send to Process Manager—**Select to allow the Process Manager to reassign or reject a request after the approval time limit has passed. The Process Manager will have option to either restart the workflow approval process or return the request to the requestor • **Auto-approve—**Allow the operation to be performed after the approval time limit has passed. • **Auto-reject—**Prevent the operation from being performed after the approval time limit has passed.
Internal auto-reject comment	(Optional) If the **Auto-reject** option was selected above, then you may type a comment that will be displayed to the requestor, such as the reason for rejection. The maximum length is 250 characters.

1.1.12.4.4.4 Specifying Actions for Custom Action Rules

When a custom action rule is triggered, its action is specified by one of the following:

- JavaScript file
- Java class file
- Template defined in the object definition for use by rules

For information about defining custom actions using JavaScript or Java classes, see Writing Automated Actions. For details about creating templates for an object definition, see Using Templates.

Specifying a Class or JavaScript File

To specify an automated action, you must upload at least one class or JavaScript file to the following folder in the Documents area:

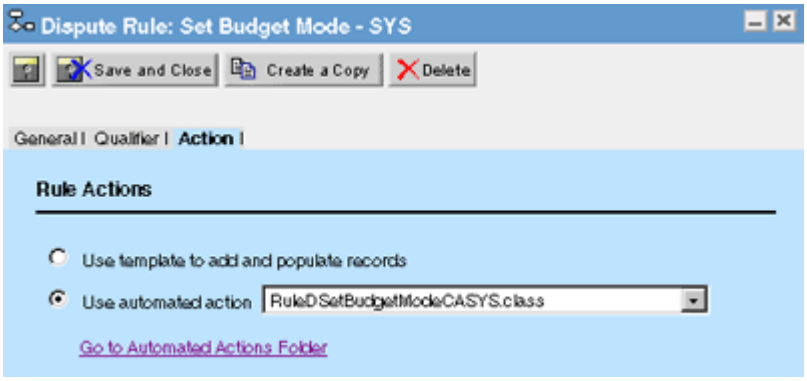
/System/Object Definitions/*objectName*/Rules/Automated Actions/

Note: *The **Action** tab of the custom action rule screen provides a link to the **Automated Actions** folder for the object definition (see the Action Tab Displaying an Error with a Custom Action image).*

To specify the Java class or JavaScript file used by a custom action rule

1. Open the **Action** tab of the custom action rule.
2. Select the **Use automated action** radio button.

   The **Use automated action** drop-down list appears.
3. From the **Use automated action** drop-down list, select the appropriate file.

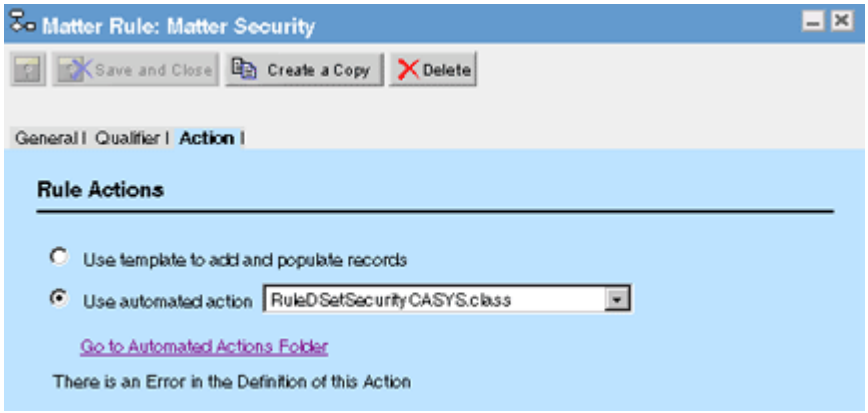


Action Tab Specifying a Custom Action

   If **No Item Selected** appears as read-only text, that means no Java class or JavaScript files have been uploaded to the Automated Actions folder for the object definition.
4. Click **Save** to associate the action with the rule.

If TeamConnect detects errors while instantiating the selected class file (in relation to the object definition and its other classes), it displays the following error on the **Action** tab:

There is an Error in the Definition of this Action



Action Tab Displaying an Error with a Custom Action, Click Link to Open Automated Actions Folder

Specifying a Template

Templates allow you to define specific fields or child records to populate when your qualifiers are met. You may access or create templates from the **Templates** tab of the object definition.

When used with templates, custom action rules have the same functionality as pre-population rules. However, the execution order of custom action rules is after pre-population, validation, and approval rules. This may be useful for some business cases, such as running approval workflow before populating a record.

To be able to specify a template for a rule, you must create at least one template for use with rules.

Note: *You cannot specify a template for a custom action rule for the* ***User*** *or the* ***Group*** *object definition.*

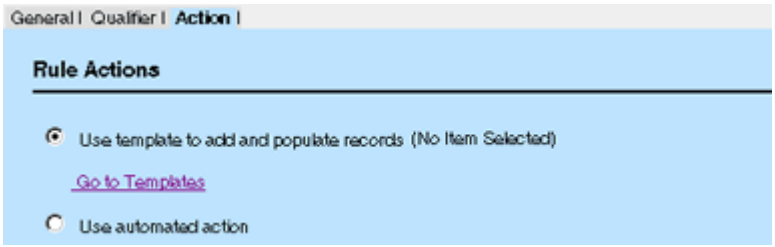
To specify the template used by a custom action rule

1. Open the **Action** tab of the custom action rule whose template you want to specify.
2. Select the **Use template to add and populate records** radio button.

   The **Use template to add and populate records** drop-down list appears.
3. From the **Use template to add and populate records** drop-down list, select the appropriate template.

   If **No Item Selected** appears as read-only text, no templates have been defined to be used with rules for the object definition.
4. Click **Save** to associate the template with the rule.

The following image shows the **Action** tab when no templates are available for the object definition. **No Item Selected** appears as read-only text and the **Go to Templates** link allows you to open the **Templates** tab of the object definition.



Action Tab of Rule Screen with No Available Templates, Click Link to Open Templates Folder

For more information about templates, see Using Templates.

To specify the template used to send notification

1. Open the **Action** tab of the custom action rule whose template you want to specify.
2. Select the **Use template to send notification** radio button.
3. From the **Use template to send notification** drop-down list, select the appropriate template.
4. From the **To/CC/BCC** drop-down list, select the type of email to be sent to the recipient: A regular email, a carbon copy email, or a blind carbon copy email.
5. From the **Recipient** drop-down list, select the location of the recipient. Notification emails will be sent to the default email address of a user or contact.
   - Select **Path** and click the **Navigator** icon to open Object Navigator, and then select an attribute. For details, see Using Object Navigator.
   - Select **Group** to send the notification to all members of a defined group.
   - Select **Address Book** to send the notification to users in an address book.
   - Select **Contact Object Collection** to send the notification to users in a defined object collection, such as All Contacts, Recently Created, Recently Modified, etc.
   - Select **Email String** to send the notification to an email address.
6. (Optional) Select the **Track in History** check-box if you want a history record to be created every time a notification is sent out. A drop-down list appears.

   From the **Location** drop-down list, select which level you want the history record to be saved in: **Record**, **Object**, or **System**. **Record** will not be available if the rule triggering event is **Delete**.
7. Click **add more** to continue adding notification recipients if necessary, or click **Save** to save the custom action rule.

Important: *Notification templates that contain an object link or object link display string will not display these items correctly in the notification email for Custom Action rules that use the Create triggering event. This is because the object link has not yet been created when the Create triggering event occurs. To avoid this issue, use the Post Commit rule type instead of the Custom Action rule type.*

1.1.12.4.4.5 Defining Settings for Scheduled Action Rules

Scheduled action rules have a set of fields on the **Action** tab where you must do the following actions:

- Select the file that is executed or the template that is used according to the schedule that you specify.
- Specify when, and how often, the action is executed after the rule has been triggered.

To specify an automated action, you must upload at least one class or JavaScript file to the following folder in the Documents area:

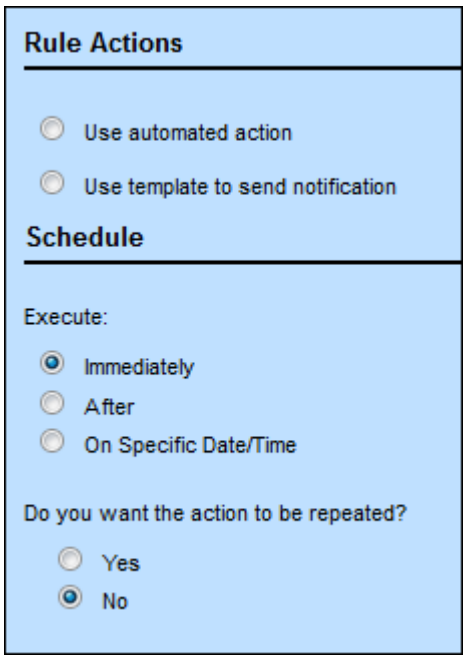
/System/Object Definitions/*objectName*/Rules/Automated Actions/

To define the action settings for a scheduled automated action rule

1. Open the **Action** tab of the scheduled action rule.
2. Select the **Use this Automated action** radio button, then select the custom action Java or JavaScript file that you want the rule to execute from the **Use this Automated action** drop-down list.

   If **No Item Selected** appears as read-only text, that means no Java class or JavaScript files have been uploaded to the Automated Actions folder for the object definition.
3. Type or select the rest of the settings as described in the Action Tab on Scheduled Action Rule Screens table.
4. Click **Save** to save the scheduled action settings as part of the rule.

For more details about scheduled action rules, see Scheduled Action Rules.



Action Tab on Scheduled Action Rule Screens

To define the action settings for a scheduled notification action rule

1. Open the **Action** tab of the scheduled action rule.
2. Select the **Use template to send notification** radio button.
3. From the **Use template to send notification** drop-down list, select the appropriate template.
4. From the **To/CC/BCC** drop-down list, select the type of email to be sent to the recipient: A regular email, a carbon copy email, or a blind carbon copy email.
5. From the **Recipient** drop-down list, select the location of the recipient. Notification emails will be sent to the default email address of a user or contact.
   - Select **Path** and click the **Navigator** icon to open Object Navigator, and then select an attribute. For details, see Using Object Navigator.
   - Select **Group** to send the notification to all members of a defined group.
   - Select **Address Book** to send the notification to users in an address book.
   - Select **Contact Object Collection** to send the notification to users in a defined object collection, such as All Contacts, Recently Created, Recently Modified, etc.
   - Select **Email String** to send the notification to an email address.
6. Click **add more** to continue adding notification recipients if necessary.
7. Type or select the rest of the settings as described in the Action Tab on Scheduled Action Rule Screens table.
8. Click **Save** to save the scheduled action settings as part of the rule.

The following table describes the fields in **Scheduled Action Rule** screens.

Action Tab on Scheduled Action Rule Screens

Field	Description
Use this Automated action	Select the Java or JavaScript file that defines the action that is executed according to the schedule. This same action is executed for each repetition that you specify in these parameters. This drop-down list contains documents that have been uploaded to **Top Level/System/ObjectDefinitions/*ObjectName*/Rules/Automated Actions** and have the document type JavaClass or JavaScriptCode. To open the folder where Automated Action files for this object definition are stored, click **Automated Actions Folder** hyperlink.
Use template to send notification	Select the email notification template that is to be sent according to the schedule. This same notification template is sent for each repetition that you specify in these parameters.

Track in History	(Optional) Select whether a history record will be created every time a notification is sent out. A drop-down list appears when the **Track in History** check-box is selected. From the **Location** drop-down list, select which level you want the history record to be saved in: **Record**, **Object**, or **System**. **Record** will not be available if the rule triggering event is **Delete**.
Execute	Specify when the automated action or notification is to be executed. • **Immediately—**The action is executed immediately after the rule is triggered. This is the default option. • **After—**The action is executed a specified period of time after the rule is triggered. Select from **Minutes**, **Hours**, **Days**, **Weeks**, **Months**, or **Years** in the drop-down list, then enter a value in the text box to specify how long after the trigger the rule will be executed. • **On Specific Date/Time—**Use the fields that appear to specify the calendar date and time for the scheduled action to be executed.
Do you want the action to be repeated?	Select whether the action is repeated. If you select **Yes**, then additional options appear, where you specify how often and how many times the action is repeated. The default option is **No**.
Repeat	Specify the period of time between rule action repetitions after the initial execution after the rule has been triggered. Additional options will appear depending on the selection in the **Repeat** drop-down list. • **Hours and Minutes—**The scheduled action will repeat after the specified number of **Hours** and **Minutes** have passed. • **Daily**—The scheduled action will repeat every day. Select the **Weekdays only** check-box to set the rule action to be repeated only from Monday through Friday. • **Weekly**—The scheduled action will repeat every week. Use the **Day** drop-down list to select on which day of the week the action will be repeated. • **Monthly**—The scheduled action will repeat every month. Use the **Day of the Month** drop-down list to select on which day of the month the action will be repeated, from **1** to **31**, or **Last** for the last day of the month. • **Yearly**—The scheduled action will repeat every year. Use the **Day and Month** drop-down lists to select on which month and day of the year the action will be repeated.
End	Specify when the rule action repetitions will stop. • **No End Date—**The scheduled action will repeat indefinitely until the rule is deleted or inactivated.

	• **After—**The scheduled action will execute for the specified number of repetitions. This is the default option, with a value of 1. • **Until**—The scheduled action will repeat until the specified date and time.

Important: *Actions that are scheduled to repeat monthly on the 29th, 30th, or 31st day of the month will not occur in months that do not have those dates. To avoid this issue, use the* ***Last*** *option for the last day of the month.*

1.1.12.4.4.6 Defining Actions for Audit Rules

When the qualifier for an audit rule is met, details about the operation that triggered the rule are automatically logged in a history record. Audit rules are designed to capture information about events, but you may define what is captured and how it is displayed.

The **Action** tab of an audit rule allows you to specify:

- A location for the history records.
- The description of the event.
- A category and custom fields to separate information about the event.

The first thing to consider when you want to audit an event is the target location to create the history record. For example, you may select a contact record as the target location to record a contact name every time it is updated. When the rule is triggered, an audit history record is created, and afterwards, you may view the contact record's **History** tab to track the events.

Next, you need to define the description for the event to be recorded. This description populates the **Description** field of the history record that is created when the rule is triggered. There is already a default description, depending on the triggering event; however, you may customize your own using a batch screen, if you need more specific information.

Finally, you may separate details about the event into specific custom fields for reporting purposes. To accomplish this, you must select a History category and populate its custom fields. When the rule is triggered, in addition to the **Description** field's contents, the custom fields that you defined are populated for the appropriate category on the **Details** block of history records.

To define the action settings for an audit rule

1. Open the **Action** tab of the audit rule.

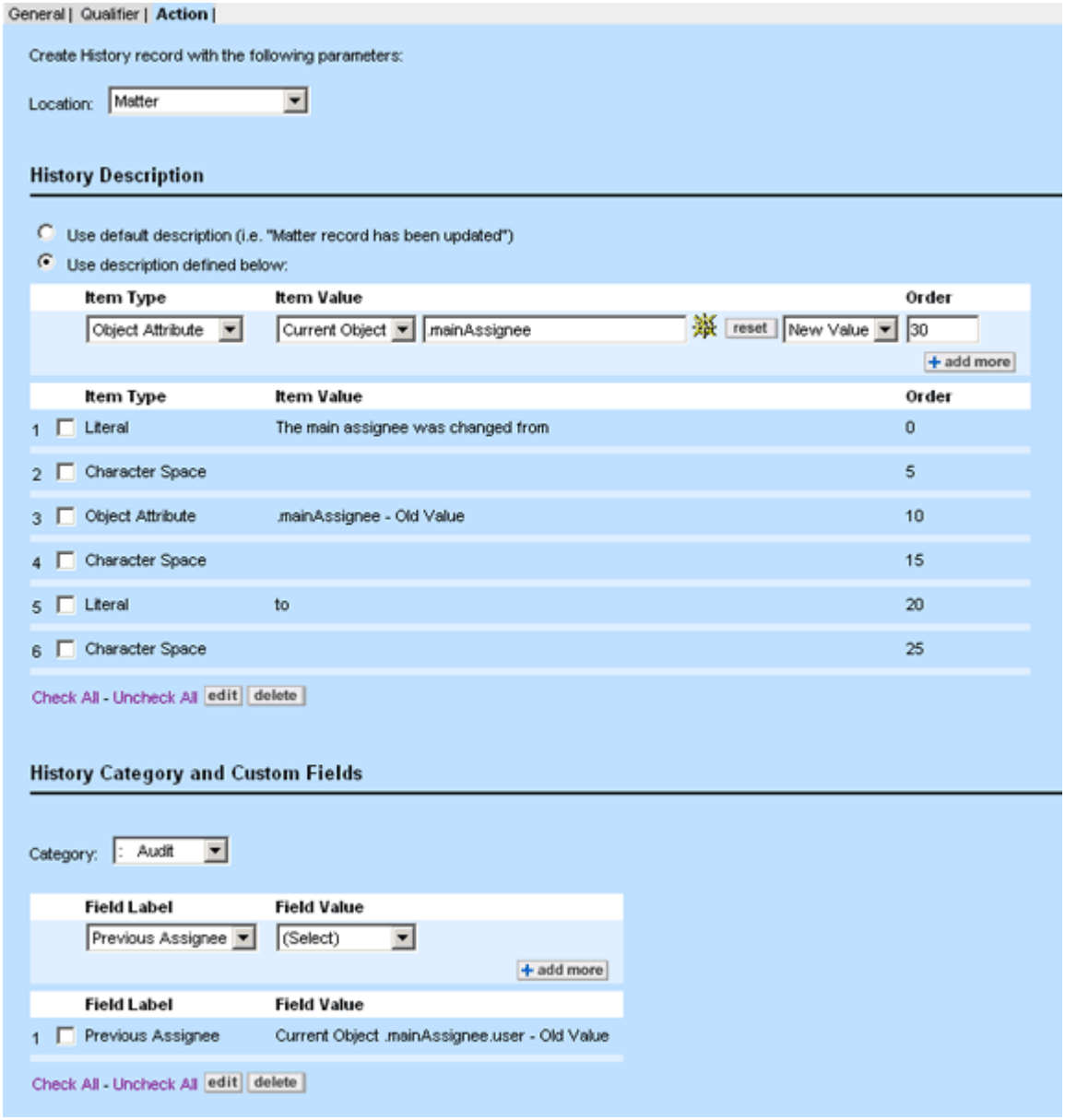
   For a description of the fields on an audit rule's **Action** tab, see the Action Tab on Audit Rule Screens table.

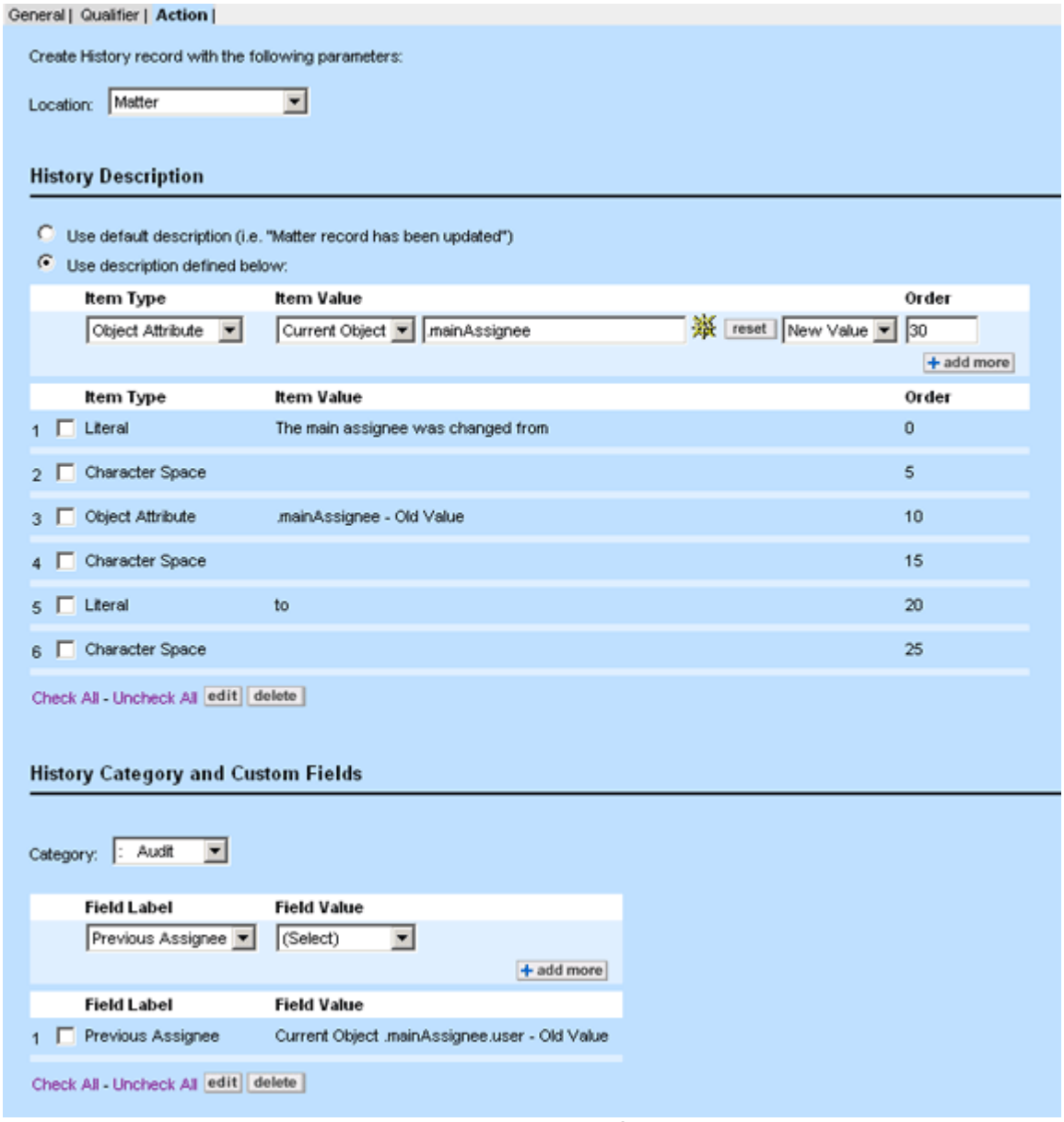
2. From the **Location** drop-down list, select a target where the history record will be created when the audit rule is triggered.

3. Define a description for the triggering event to populate the **Description** field of the history record.

4. (Optional) From the **Category** drop-down list, select a History category and define its custom fields to capture event details.
5. Click **Save** to save the settings as part of the audit rule.

The following images show the **Action** tab of an audit rule screen. The action settings for the rule in this image creates a history record in the affected matter record with a custom description that states, for example, "The main assignee was changed from Jake Ashby to Larry Finch." It will also populate a custom field called "Previous Assignee" in the Audit category on the **Details** block of the history record.





Action Tab on Audit Rule Screens

The following table describes the items on the **Action** tab.

Action Tab on Audit Rule Screens

Field/Button	Description
Location	Select the target location for storing the history record that is created when the audit rule is triggered. Depending on the object that you are defining, this drop-down list may include some or all of the following options:

	• **Current record**—For example, if you select Matter, a related history record will be created in the affected Matter record. • **Object definition**—For example, if you select Matter Object Definition, a history record will be created in the Matter custom object definition. • **Parent project or projects**—For example, if you select a Claim's parent project, Policy, a related history record will be created in the policy record. • **System**—If you select this item, the history record will be created in an area reserved for the TeamConnect System user. You may access history records assigned to this location on the **History** tab of the **System Settings** screen.
Use default description (i.e. "Matter record has been updated")	Select this option if you want TeamConnect to insert a default description of the event that triggers the audit rule in the history record's **Description** field. The default description combines the object type, record name, and triggering event to form a description such as "Matter Simmons vs. Acme Auto Rental has been updated."
Use description defined below	Select this option if you want TeamConnect to insert a custom description of the event that triggers the audit rule in the history record's **Description** field. This is the default option, and you must use the associated batch screen to define a custom description. For more information on whether you need a default description or a custom description, see Audit Default Description.
Item Type	Select an item from this list to build the description for the history record. Depending on the option that you select in this field, different settings appear for the Item Value: • **Object Attribute**—Select this option to define an object attribute. • **Literal**—Select this option to manually enter a text value. • **Line Break**—Select this option to insert a carriage return in the text description. • **Character Space**—Select this option to insert a space between items in the text description.
Item Value	Define a value to build the description for the history record.

	• For object attributes, select **Current Object** or **Current User**, click the **Navigator** icon to open Object Navigator, and then select an attribute. For details, see Using Object Navigator. • For literals, enter a custom message in the text field.
Old Value or New Value	Select **Old Value** or **New Value** for the attribute. This drop-down list only appears if the **Item Type** is an object attribute. The old value refers to the attribute's value in the TeamConnect database before the record is changed and updated by an action. The new value refers to the attribute's value after the record has been updated and saved. ***Tip:*** *This may be useful in building the history record's description by capturing information that might not exist in the record after it is modified and saved.*
Order	Enter an integer to indicate the Order of the item you defined to place it correctly within the entire description statement.
Item List	Displays the items that are already added to be used in the custom description. Select the check-box of an item and click **edit** or **delete** to make changes.
Category	Select a category from this list to be automatically added to the history record. This list shows all the categories available in the History object definition. Make sure that you have created descriptive categories in the History object definition and defined custom fields within these categories that you plan to populate with the audit rule. For details, see Configuring History Object Definition for Audit Tracking. If you do not select a different category, the **Root** category is automatically used. After you select a category, you may use the **Field Label** and **Field Value** fields to define the content that will populate the category's custom fields.
Field Label	Select the label of the custom field that you want to use in the history record.

	This drop-down list contains the labels for the custom fields in the History object definition. The values available in this list depend on the History category selected in the **Category** field. These field labels appear on the **Details** block of the history record that is created when the rule is triggered. This field is not available for the **User** and **Group** object definitions.
Field Value	Select **Current Object** or *Current User*, click the **Navigator** icon to open Object Navigator, and then select an attribute to populate the field that is identified in **Field Label**. For details, see Using Object Navigator. You must select whether you want the field to be populated with the **Old Value** or **New Value**. Make sure you have defined any necessary custom fields for the category you want to use in the History object definition. Make sure to match the field type to the attribute type, such as matching a date field to an object attribute that stores a date value. This field is not available for the **User** and **Group** object definitions.

1.1.12.4.5 Rule Administration

When you have finished defining all of the necessary rule specifications for your system, you must perform the actual administration of rules in Designer. This involves tasks such as creating, deactivating, modifying, deleting, and troubleshooting rules. This section describes these procedures in detail so that you may work with your own rules in TeamConnect.

1.1.12.4.5.1 Defining User Interface Rules

The following procedure describes the process of creating rules with predefined actions within TeamConnect. Security, validation, and approval rules may be created using these instructions. To create rules using Java class files or JavaScript for their qualifiers or actions, see Creating Automated Qualifiers and Actions.

Before creating a rule, be sure that you create the correct rule type. For descriptions of security, validation and approval rules, see Rule Types.

To create a rule

1. In the Designer window, from the **Go to** drop-down list select **Object Definitions**.
2. Select the appropriate object definition and then click the **Rules** tab.
3. Click **New**.

4. The corresponding **General** tab appears with blank fields.
5. Set the **General Rule Information**. For details, see Setting General Rule Information.
6. Click the **Qualifier** tab and create all of the necessary qualifier conditions. For details, see Rule Qualifiers.
7. Click the **Action** tab and define all of the necessary Rule Action information. For details, see Rule Actions.
8. Click the **General** tab.
9. Select the **This rule is Active** check-box to activate the rule.
10. Click **Save**.

The rule is immediately active and is triggered when a user attempts to perform the operation that is selected as the trigger.

1.1.12.4.5.2 Deactivating Rules

You may easily deactivate rules that are currently active. For example, you may want to temporarily deactivate a rule so that it does not interfere with the behavior of a particular rule that you are testing. You may also want to deactivate a rule because you are replacing it with a different rule.

To deactivate a rule

1. Open the rule that you want to deactivate.

   For details about accessing rules, see Accessing the Rules Screen.
2. On the General tab, clear the **This Rule is Active** check-box.
3. Click **Save**.

   The rule has been deactivated and will no longer be triggered. However, it still appears in the list of rules for the Object Definition.

Points To Remember

When deactivating a rule, consider the following points:

- For all rule types, as soon as the rule is deactivated, it ceases to be triggered on the triggering events. For example, when users click **Save** on a new record, a deactivated rule with a **Create** trigger is not triggered.
- When you deactivate a rule, you may re-activate it later as needed. It remains in the rule list for the object definition.
- When you deactivate an approval rule, approval processes that have already been started are not affected when you deactivate the rule.
- When you deactivate a scheduled action rule, actions that have already been scheduled by the rule are not affected when you deactivate the rule. If you want to cancel actions that have already been scheduled, you must delete them using the Scheduled Actions Monitor.

1.1.12.4.5.3 Modifying Rules

You may modify an existing rule when necessary. If the rule is active, the changes immediately take effect and may be observed the next time the rule is triggered. If the rule is inactive, the changes may be observed when the rule has been activated and then triggered.

You may modify existing rules for any of the following reasons:

- To modify existing qualifier items.
- To add or delete qualifier items.
- To change the message when the rule is triggered.
- To change the order in which the rule is executed.
- To change the rule trigger(s).
- To change the route for an approval rule to a different route, or modify other information on the approval rule's **Action** tab.

  Approval processes that have already been started are not affected if you modify the approval rule's action settings or select a different route for the rule.
- To change the scheduling of a scheduled action rule.

  Actions that have already been scheduled are not affected by modifications to any of the rule's action settings. The changes only affect actions that are scheduled by the rule after you save the changes to the rule.
- To change the file being used as the action for a custom action or scheduled action rule.

  Actions that have already been scheduled by a scheduled action rule are not affected if you select a different file for the rule's action. The change only affects actions that are scheduled by the rule after you select the new file for the rule.
- To deactivate the rule (see Deactivating Rules).

Note: *Some modifications to rules may require you to click the* ***Refresh*** *button to see the modified information reflected in the list of rules.*

1.1.12.4.5.4 Deleting Rules

You may need to delete a rule during the design phase of your TeamConnect implementation, or because you are sure that the rule is not necessary.

You may delete active and inactive rules. When you delete an active rule, the rule immediately stops taking effect. Users are able to perform the corresponding operation, such as creating, modifying or posting records, without the previous constraints of the rule.

However, rules that have already been triggered are not affected when you delete a rule. For example, if you delete an approval rule that has already been triggered and the approval request has already been routed, then the approval process is not affected by the deletion of the rule.

Tip: *If you want to keep the rule for future use or future reference, but you do not want the rule to continue to be triggered, you may deactivate the rule. For details, see Deactivating Rules.*

To delete a rule

1. Click the **Rules** tab of the appropriate object definition.
2. Click the name of the rule you want to delete.

   The **General** tab of the rule opens by default.
3. Click **Delete** at the top of the screen and confirm that you want to delete the rule.

The rule has been deleted and will no longer be triggered by user operations.

1.1.12.4.6 Troubleshooting Rules

To assist you with creating rules, this table provides a quick troubleshooting guide of common problems and their solutions.

Rule Troubleshooting Tips

Possible problem	Possible solution
An error message appears for a rule that must not be triggered in the situation you are testing.	Check for the following issues: • Open the error messages of the rules being triggered. • Do they have the correct messages? • Does the rule have multiple triggers associated with it? Are those the correct triggers? ***Note:*** *You may have another rule being triggered with the same error message, especially if you used the* ***Create Copy*** *button to create this rule.*
A rule is not being triggered.	Once you create any rules, test them to see whether modifications are needed to get a rule to trigger in the correct situations. If your rule is not being triggered, check for the following issues: • Is the rule active? • Did you select the correct rule type? Security and validation rules are easy to confuse. • Is the qualifier defined correctly? • Is the entire qualifier met? If at least one of the qualifier conditions or the logic that you have defined is not met, the rule is not executed.
My Approval rule is not being triggered at the right time.	Check for the following issues: Is the rule active?

	Is a route selected? Does the route have members? ***Note:*** *If a group is the only stop member in the selected route and there are no active users added to the group, there is no indication that the rule was triggered, because the approval request was not sent to any users.* • Is the qualifier defined correctly? • Is the entire qualifier met? If at least one of the qualifier conditions or the logic that you have defined is not met, the rule is not executed.

1.1.12.5 Creating Routes

A *route* is a sequence of users whose approval is required for users to perform a certain action. Routes are required for approval rules. For details about how approval rules are constructed, see Approval Rules.

Routes may have multiple *stops*, which define the order in which the approval is sent to each user whose approval is required. The members of a route may approve or reject the attempted operation, reassign it to another user, or even edit items in a workflow that are still pending approval. ***Note: Editing items pending approval in workflow is only available in TC 4.0.8 and later.***

Members of a route may include:

- User groups
- Users who have a certain role in a project record
- Other users who are identified by association

  If a route is associated with all objects, only the first choice (user groups) is available when creating stops. The choices for users with roles, or users identified by association, are available for routes that are linked to one specific object type.

If the sequence of approval is important, the stops must be separate. Otherwise, all members may be in the same stop.

For example, to create a route that requires approval from a transaction attorney and a supervisor before an invoice payment over $10,000 is posted, you would need an approval rule that checks whether an invoice being posted is for more than $10,000. If only one person must approve the posting, then the route would need only one stop with all members.

If, however, the invoice may only be posted after a transaction attorney and a supervisor both approve the posting, in that order, then the route would contain two stops. The first stop would send the approval to the transaction attorney. If the transaction attorney approves the posting, the approval would then be sent to the supervisor.

You may also define multiple branches (with the same rule) for the same operation when the levels of authority to approve differ depending on the qualifiers that trigger the approval rule. This is because it

is impossible to have an approval finish part of the way through the route. It must finish the route before it is approved.

For example, for invoice posting approval:

- **Up to $5,000:** the Main Assignee's approval is sufficient.
- **More than $5,000, but less than $20,000:** the Main Assignee and the supervisor must approve the posting.
- **More than $20,000**: the first two individuals and the vice president must all approve the posting.

1.1.12.5.1 Accessing Routes Screen

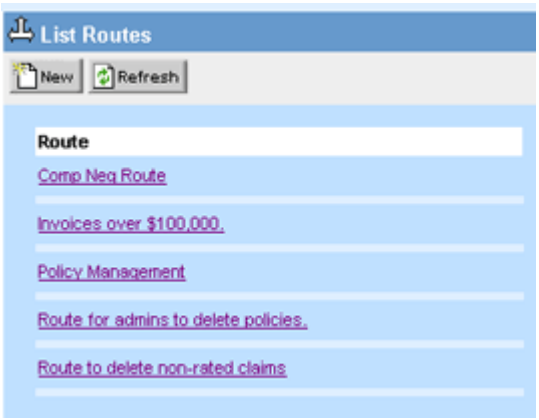
If you have user group rights to Routes, you may access them from the **Setup** link, which starts Designer.

To open a new or existing route definition

1. In the Designer window bar, in the **Go To** drop-down list, select **Routes**.

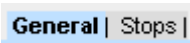
   The **List Routes** screen appears.
2. Do one of the following actions:
   - In the list of existing routes, click the name of the route you want to open.
   - Click **New** to create a route.

     The **General** screen displays by default.
3. Click the **Stops** tab to view the stops information, if necessary.



Routes Screen

You may create, view, and modify a route on its **Route** screen, which has the following tabs:

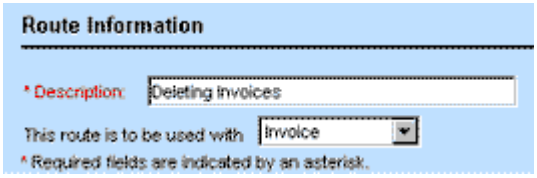


- **General**—Displays the name of the route and object for which it may be used. See Defining General Route Information.

- **Stops**—Displays the stops and members of each stop. See Route Stops.

1.1.12.5.2 Defining General Route Information

General route information includes the name of the route and the object to which it applies. These fields are located on the **General** tab of the route.



General Tab on Route Screens

The following table describes the items in the **General** tab of **Route** screens.

General Tab on Route Screens

Field	Description
Description	Enter a name (maximum 250 characters) that uniquely identifies the route. This description appears on the **Action** tab of an approval rule to allow you to select it.
This route is to be used with	Select the object for which the route may be used. You may either specify one object, or select **(All)** to make the route accessible to all objects. • If the route you are creating is only used for one object, such as Expense, then select the name of that object in the list. You must select a specific object if you want to add a **User with role** or **User path** as a stop member (see Member Types). • If you want to be able to use the same route of approvers for more than one object, you must select **(All)**. For example, you may need to use the same route for both task and expense approval rules. If the route is available for all objects, then you may only add specific users or groups to the route. You are unable to add a user with role or user path as a stop member (see Member Types). When the route is saved, you cannot change this selection.

After saving the name and the object selection for the route, you may click the **Stops** tab and define the stops in the route.

1.1.12.5.3 Route Stops

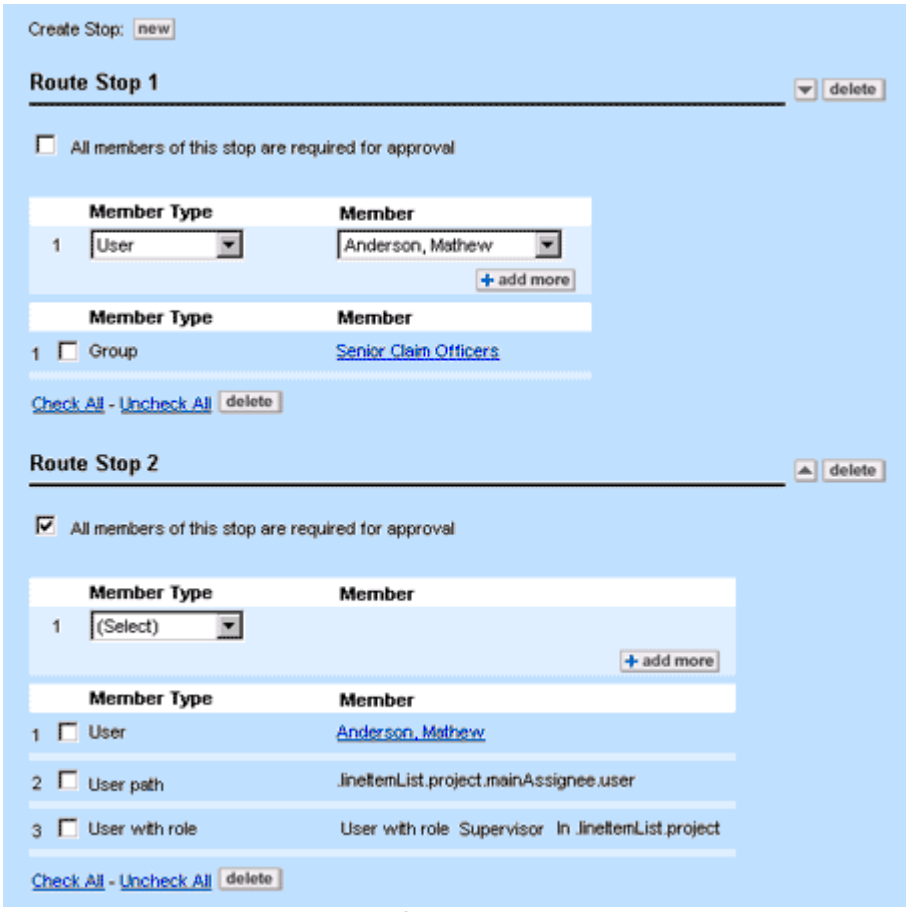
Stops represent approval rule check points. The route follows the order in which you organize the stops, starting with Stop 1. A stop consists of members who may approve or reject the operation

attempted by a user. All members of each stop receive the approval notice in their **My Approvals** screen.

Points To Remember

- If all users in a stop do not exist, the stop is skipped and the route proceeds to the next stop. For example:
    - If a stop includes only the Main Assignee of a Claim and there is no Main Assignee, then the stop is skipped and the route proceeds to the next stop.
    - If a stop includes only a certain assignee in the parent project of a project that is being deleted. If there is no parent project, then the route proceeds to the next stop.
    - If a user with a specified role is assigned to a stop in a route, and there is no such role assigned to any active assignee or group, the assignee is skipped and the request goes to the next approver.
- If an assigned user with a specified role exists, but the user account is inactive, an error message appears if all members are required for approval or if there are no other valid members in the stop.
- Rejection of an approval request always works on a "first come, first serve" basis. Once any designated approver rejects a request, the request record is removed from the **My Approvals** list of all route members, regardless of which stop in the route the rejection occurred.
- Approval may work on a "first come, first serve" basis only if you want it to. For each stop of the route, you may control whether one approval is enough to allow the operation, or all stop members must approve the operation before the route proceeds to the next stop.
- If the sequence of approval does not matter, then you must place all of these members in one stop. Placing multiple members in one stop and requiring approval from all stop members (selecting the check-box) allows you to give more time for all members to approve the operation, rather than having them wait for the previous stop members to approve the operation first.
- If the sequence of approval does matter, then each member whose approval is required must be placed in a separate stop, in the order in which their approval is required. You must also make sure that the expiration time allows for all stops to be reached.

Stops may be viewed, added, deleted, or modified on the **Stops** tab of the route. The **Stops** tab also defines the order of the stops and whether all members of each stop must approve the request before the route proceeds to the next stop.



Stops Tab

The following table describes the items on the **Stops** tab.

Stops Tab

Field or button	Description
new	Click to create a new stop for the route. The new stop is added to the end of the route.
All members of this stop are required for approval	Select this check-box if all members of the stop are required to approve the request. If the check-box is selected, then the route does not continue to the next stop until all users in the stop have approved the operation. If this check-box is NOT selected, then the route continues to the next stop as soon as one user in the stop has approved the operation. This check-box requires all members no matter how they are added to the stop. This means that a member's approval is required in all of the following situations:

	• More than one member (of any type) is added to the stop. Each member's approval is required. • A **Group** is a member of the stop. This check-box requires an approval from all members of the group. • A **User with role** or **User path** stop member points to more than one user with the specified role. For example, there are two General Counsel assignees (user with role). Approval is required from each assignee with the specified role. Whether or not this option is selected, a single user may reject the attempted operation and thus delete the request from the **My Approvals** screen of all route members. You cannot set a route to reject an operation only after all members have rejected it.
Member Type	Select the type of stop member you want to add to the stop. For details, see Adding Stop Members.
Member	Select or identify the user or group you want to add to the stop. For details, see the following: • Adding Groups as Stop Members • Adding User Paths as Stop Members • Adding Users With Roles as Stop Members
Member List	Displays the members of the corresponding stop.
Move Up or Down Arrows	Use these arrows to change the order of the stops if needed. Click the respective arrow to move the entire stop up or down one level. The route follows the order in which you organize the stops, starting with **Stop 1**.
delete	Click to delete the corresponding stop from the route. Each delete button corresponds to the stop with which it is aligned. ***Note:*** *Do not confuse the delete buttons for the stops with the delete button at the bottom of the screen that appears in all batch screens. The delete button for the batch screen applies to stop members rather than an entire stop.*
Visit stops in sequence	Select to specify that the route visits each stop in numerical order. This option does not appear if you can use the route with all record types.

Custom sequence	Select to place a condition on a stop to determine whether the route visits the stop. Refer to the Custom Sequence Entries table for details about this field. This option does not appear if you can use the route with all record types.

1.1.12.5.3.1 Adding Stop Members

You may add stop members on the **Stops** tab of the route. You may add static members and dynamic members to a stop, depending on your needs.

Important: *Make sure that there is at least one active user in a route that is being referenced by an approval rule. If no users are identified in a route, the route behaves as if everyone in the route approved the operation. The operation will be completed.*

Adding dynamic stop members requires an understanding of Object Navigator. For details, see Using Object Navigator.

Member Types

The Member Type determines how users are included in a route stop as members: as members of a static user group, or as dynamically selected members. The following table explains the three member types.

Stop Member Types in Routes

Member type	Description	Example
Group	• Allows you to select a specific group to add as a stop member. The request is sent to all active users who are part of the group. • This type of stop member is static. It identifies one group by name.	The group Senior Claim Officers must approve posting of invoice payments over $100,000.
User path (Not available in routes that are specified for **All** objects in the route's **General** tab.)	• Allows you to add a stop member based on how they are associated to the object record. For example, a user path could identify the user who created a record, the user who is the main assignee, and so on. • This type of stop member is dynamic. It identifies a user (or, in some cases, multiple users) based	Posting of invoice payments over $100,000 must be approved by the user who is the main assignee of the claim that is identified in the line item of the invoice. In this example, if more than one project is listed in the line items of the invoice, then the approval goes to the main assignee of each project.

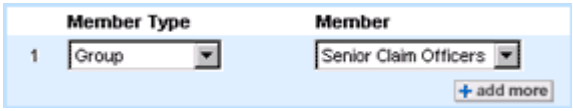
	on the user's relationship to the record. • A user who is not active does not receive the approval request.	
User with role (Not available in routes that are specified for **All** objects in the route's **General** tab.)	• Allows you to add a stop member based on the user's assignee role in a project. • If a project has more than one assignee with the same role, the approval request is sent to each user who has the selected role. • This type of stop member is dynamic. It identifies a user (or, in some cases, multiple users) based on the user's relationship to the record. • An assignee who is not active does not receive the approval request. • An assignee whose underlying user is inactive does not receive the request.	Posting of invoice payments over $100,000 must be approved by the user who is the Supervisor of the claim. In this example, if the claim record has more than one supervisor, then the approval goes to each supervisor.

The steps necessary to add members to a stop are different for each member type. The following are step-by-step instructions to help guide you through adding stop members:

- Adding Groups as Stop Members
- Adding User Paths as Stop Members
- Adding Users With Roles as Stop Members

You may add user groups to route stops. If any or all members of a certain group must always approve certain actions, then adding the group as a member of the route is a good idea.

Even though a group is a static stop member type, the route automatically uses only the current, active members of the group in the approval process.



Groups as Route Stop Members

To add a group as a stop member

1. Click the **Stops** tab of the route to which you want to add a group as a member.
2. From the **Member Type** drop-down list, select **Group**.

3. From the list of groups, select the appropriate group.
4. If appropriate, select the check-box requiring approval for all members.

   This requires approval for all members of the group and all other members of the stop, if there are other members.
5. Click **add more** to continue adding stop members, or click **Save** to save the route.

A **User path** stop member identifies a dynamic user. The user who receives the approval depends on which user is identified by the path. This stop member type is only available if a specific object is identified for the route on its **General** tab.

To add a user path as a stop member

1. Click the **Stops** tab of the route to which you want to add a user path as a stop member.
2. From the **Member Type** drop-down list, select **User path**.

User Paths as Route Stop Members

3. Click the **Navigator** icon and create a path using Object Navigator.
4. If appropriate, select the check-box to require all members of the stop to approve the requested operation.

   This requires approval from all members of the stop, including all users who are identified by this path, as well as other members of the stop.
5. Click **add more** to continue adding stop members, or click **Save** to save the route.

When adding a user path as a member of a stop, keep in mind that if there is no user identified by the path, then the member is not added to the stop. For example, if you want the route to go to the Main Assignee of a project and this is the only member of the stop, then the stop is skipped if there is no Main Assignee, and the route automatically proceeds to the next stop.

When creating a path to a user for a member of a stop, the path must end with an attribute in the object model that identifies a user. This means that your final selection in Object Navigator must be one of the following:

- `user->`
- `createdBy->`
- `modifiedBy->`
- `expensedBy->`
- `enteredBy->`
- `transitionedBy->`
- Another attribute that identifies a user.

Important: *Only users may be members of a route. Do not select an attribute that identifies a contact as your final selection for a user path.*

The following table provides sample user paths created for a stop member.

User Path Examples for Route Stops

User identified by this path	What to select in Object Navigator to build this path
The user who created the record.	`createdBy->(ok)`
The user who is the Main Assignee of the project for which you are creating a route.	`mainAssignee->(traverse)` `user->(ok)`

The user who is the Main Assignee of the project to which the current object for which you are creating a route (such as Account, Expense, Task, etc.) is a related object.	`project->(traverse)`
	`mainAssignee->(traverse)`
	`user->(ok)`

The user who is the main assignee of the project indicated in the invoice's line items (for a route created for the Invoice object).	`lineItemList->(traverse)`
	`project->(traverse)`
	`mainAssignee->(traverse)`
	`user->(ok)`

The user who entered or posted the expense (for a route created for the Expense object).	`expensedBy->(ok)`

A **User with role** stop member identifies a dynamic user who is an assignee of a project. The user who receives the approval depends on which user is identified by the Assignee role indicated in the stop. This stop member type is only available if one specific object is identified for the route in its **General** tab.

To add a user with role as a stop member

1. Click the **Stops** tab of the route to which you want to add a user with role as a stop member.
2. From the **Member Type** drop-down list, select **User with role**.

Users with Specific Roles as Route Stop Members

3. Select the appropriate role in the second field.
4. Do one of the following actions:

- If the role you selected belongs to the object for which you are creating the route, you do not need to create a path. Instead, continue to the following step.
- If the role you selected belongs to a different object than the object for which you are creating the route, click the **Navigator** icon and create a path to the necessary project using Object Navigator.

  For example, if you want the stop member to be the user with the role GC Attorney in the parent project of the project for which you are creating the route, select parent-> in Object Navigator.

5. Click **add more** to continue adding stop members, or click **Save** to save the route.

When you create a user with role stop member, follow these guidelines:

- If you are creating a user with role stop member and the Assignee role exists within the object for which you are creating the route, you do not need to create a path to identify the role.
- If the role belongs to a different object, you must create a path that indicates which object the role is from, in relation to the object for which you are creating the route, as shown in the examples in the User with Role Examples for Route Stops table.

When creating a path to a user with role, the path must end with an attribute in the object model that identifies a project (custom object). This means that your final selection in Object Navigator must be one of the following:

- `project->`
- `parent->`
- `applProject->`
- `leftProject->` (this is not common in routes)
- `rightProject->` (this is not common in routes)
- Another attribute that identifies a project.

Important: *To build a path that identifies the PROJECT in which the user has the specified assignee role, do not select an end-of-path attribute that identifies the user.*

The following table provides sample **User with role** paths for stop members.

User with Role Examples for Route Stops

Project identified by this path	Example	What to select in Object Navigator to build this path
The parent project of the current project. ***Note:*** *This path may only be used for custom objects that*	When creating a route for Claim (a custom object): If Policy is the parent object of Claim, this would send the approval request to assignees of the policy	`parent->(ok)`

have parent-child relationships.	record that is the parent of the claim record.	
The project to which the current object (such as Task or Expense) is related.	When creating a route for Task, Expense, or other system object that may be related to a project record: This would send the approval request to assignees of the project that is selected in the **General** information for the record.	`project->(ok)`
The project to which an account belongs.	When creating a route for account: This would send the approval request to assignees of the project that is selected in the **General** information for the account.	`applProject->(ok)`
The project referenced on a line item of an Invoice. If more than one project is listed, then the approval goes to each user who has the selected role in each of the projects.	When creating a route for invoice: This would send the approval request to assignees of the projects that are referenced by each line item on the invoice. If all line items in the invoice reference the same project record, then only the assignees of that project receive the approval request.	`lineItemList->(traverse)` `project->(ok)`
The parent of the parent project of the current project.	When creating a route for litigation (a custom object): In a litigation record that has a parent claim, which in turn has a parent policy, this would send the approval request to the assignees of the policy record.	`parent->(traverse)` `parent->(ok)`

When adding a user with role as a member of a stop, keep in mind that if there is no user identified by the path, or if the user is inactive, then the member is not added to the stop. For example, if you want the route to go to the General Counsel assignee of a project and this is the only member of the stop, then the stop is skipped if there is no such assignee, and the route automatically proceeds to the next stop. If there are no further stops, then the request is approved.

Tip: *You may add all project assignees as members of a stop by adding a* ***User with role*** *member for each role that is possible for the project. TeamConnect automatically routes the approval request to all users who are assignees of the project.*

1.1.12.5.3.2 Customizing Route Stops

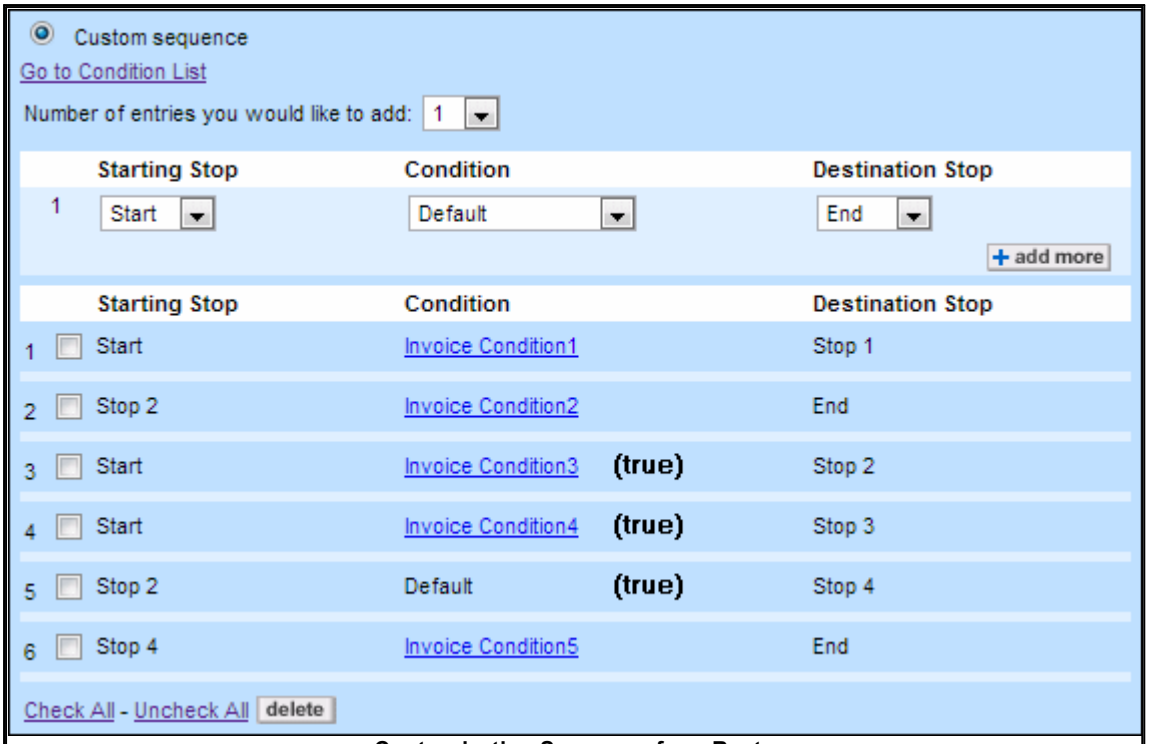
When you add stops to a route, records visit each stop by default. If you want a record to visit stops when certain conditions are present, you can customize the route.

Select **Custom Sequence** on the **Stops** tab of a route to enter the custom sequence. The following table describes the fields for entering a custom sequence.

Custom Sequence Entries

Column Name	Description of Fields	How Fields Work
Starting Stop	Specifies where the route starts or the last stop that just occurred. You can select the following fields: • **Start**—To indicate the condition that a route evaluates at the beginning of the workflow process. • Stops in route—To indicate the condition that a route evaluates after the specified stop is complete.	If more than one entry has the same field in the first column, the route evaluates the first of those entries. The route continues to each entry with the same field under **Starting Stop** until a condition is true. If no conditions are true, the workflow process ends. ***Note:*** *When a workflow process ends, the action that put the record into workflow occurs.*
Condition	Specifies the conditions available for the record type of the route. You can select the following fields: • **Default**—To indicate that a route does not have to evaluate a condition before visiting the **Destination Stop**. • Condition name—To indicate the condition that must be true for the route to visit the **Destination Stop**.	When evaluating a condition, the route visits the **Destination Stop** for the conditions that are true. See Conditional Expressions for more information about conditions.
Destination Stop	Specifies the stop that the route visits if the condition is true. You can select the following fields: • Stops in route—To indicate the stop the route visits if the condition is true. • **End**—To indicate that the workflow process ends if the condition is true.	For each entry, the **Destination Stop** number must come after the **Starting Stop** number.

In the following image, consider an example of how the route evaluates each entry for a record. In this example, the record visits only Stop 2 and Stop 4 before completing. **Invoice Condition3** and **Invoice Condition4** are true for the record and all other conditions are false.



Customization Sequence for a Route

The following actions occur for the example:

1. From the first entry with **Start** as the **Starting Stop** (line 1), the route evaluates **Invoice Condition1**. Because **Invoice Condition1** is not true, the route continues to the next **Start** entry (line 3).
2. From the second entry with **Start** as the **Starting Stop** (line 3), the route evaluates **Invoice Condition3**. Because **Invoice Condition3** is true, the route visits Stop 2.

Note: *Even though* ***Invoice Condition4*** *is true, the route does not evaluate the condition because the previous* ***Start*** *entry is true.*

3. After Stop 2 is complete, from the first **Stop 2** entry (line 2), the route evaluates **Invoice Condition2**. Because **Invoice Condition2** is not true, the route continues to the next **Stop 2** entry (line 5).
4. The second **Stop 2** entry (line 5) lists **Default** as the **Condition**, which means that the route visits Stop 4 without evaluating a condition.
5. After Stop 4 is complete, from the only **Stop 4** entry (line 6), the route evaluates **Invoice Condition5**. Even though **Invoice Condition5** is not true, the workflow process still ends and can no longer send requests to approvers because no more entries with **Stop 4** exist in the **Starting Stop** column.

To customize the sequence conditions

1. From the route, click the **Stops** tab.
2. Add all stops to the page. See Adding Stop Members for information about adding stops.
3. Select the **Custom Sequence** option, as shown in the Customization Sequence for a Route image.
4. From the **Starting Stop** drop-down list, select **Start** or a stop. **Start** indicates which entry the route evaluates first before visiting a stop. A stop indicates which entry the route evaluates after that stop is complete.
5. From the **Condition** drop-down list, select the condition that must be true for the record to continue to the **Destination Stop**.
6. From the **Destination Stop** drop-down list, select the stop that the route visits if the condition is true.
7. Click **add more** to save the entry.
8. Add additional conditions if necessary.
9. Save the route.

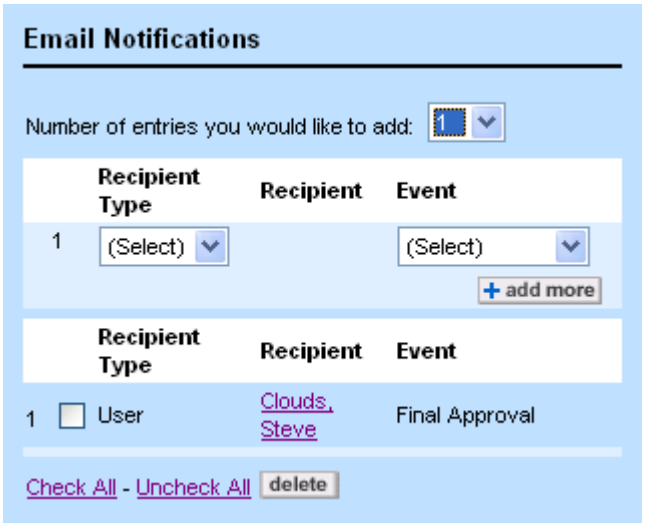
1.1.12.5.4 Adding Email Notification Recipients

You may configure email notifications to dynamically selected users or groups for the following events related to a pending approval request:

- Final approval
- Final rejection
- Error

To add a user or group as an email notification recipient

1. Click the **Email Notifications** tab of the route to which you want to add an email notification recipient.



Email Notifications Tab

2. From the **Number of entries you want to add** drop-down list, select the number of recipients to add.
3. From the **Recipient Type** drop-down list, select **Group**, **User Path**, or **User with Role**.
4. From the resulting **Recipient** drop-down list, select either the user name or group name.
5. From the **Event** drop-down list, select one of the following:
   - Final approval
   - Final rejection
   - Error
6. Repeat steps 3-5 for adding multiple recipients.
7. Click **add more**.
8. Click **Save**.

Note: *When a user is the only recipient of an email notification, it is sent in the user's preferred language. When there are multiple recipients, the email is sent in the selected system language.*

The following table describes the items in the **Email Notifications** tab.

Email Notifications Tab

Field or button	Description
Number of entries you want to add	Select the number of users or groups to add to the email notifications recipients list.

Recipient Type	Select **Group**, **User Path**, or **User With Role**.
Recipient	Select either the group name or a dynamic user. Make this selection using the steps previously described in: • Adding Groups as Stop Members • Adding User Paths as Stop Members • Adding Users With Roles as Stop Members
Event	Select one of the following: • Final approval—Sends an email notification when a request has been approved by the final approver/stop members. • Final rejection—Sends an email notification when a request has been rejected at the first stop • Error—Sends an email notification.
Email notification recipients list	Displays the current route's list of email notification recipients.
delete	Select the current email notification recipient box(es) and click the **delete** button to remove that recipient from the notifications list.

Important: *To set email notifications for users involved with the approval process as well as users with no direct role in the approval process, email notifications must be properly configured.*

1.1.12.6 Requirements for Rule Specifications

Before writing your custom code, prepare the following information:

- A description of the system design with all object definitions, their custom fields, blocks, object views, and so on.
- A complete list of user groups and their rights.
- A complete list of business rules that need to be implemented.

For each rule in the rule specification documents, a minimum of the following requirements must be specified:

- Object definition for which the rule must be created and defined.
- Rule description/title that will be used when defining the rule.
- Rule type (for a complete list, see the Rule Types table).

- Trigger (for a complete list, see the Rule Triggers table).
- Qualifier/condition:
  - Whether it is done through the user interface or API.
  - If through the API, are you choosing Java or JavaScript?
  - Whether it has parameters or not. If, yes, which ones?
- Action:
  - Whether it denies, approves, or performs a custom action.
  - If the action is to deny, the message that appears to the user.
  - If the action is custom, the use of Java or JavaScript.
  - Whether it has parameters or not. If, yes, the parameters needed.
- List of fields used in the rule, with their names (not labels), full tree positions (for custom fields), full lookup item tree positions (for custom fields of type List), full category tree positions under which custom fields are created.

Consider the following when reviewing rule specifications:

- The rule does not duplicate or replace rights set in the user or group accounts (functional level security).

  For example, if a rule is to prevent a user from creating, deleting, or updating records of a specific object definition, determine whether this may be done through the rights set at the user or group account level. You may also use record-level security to assign rights to users.

- Whether the indicated rule type and action match.

  For example, if the action is to deny and display a message, use a validation rule instead.

- Whether the trigger is identified properly.

  This is important because the exact appropriate behavior must be identified. For example, in rules, there is no such trigger as Save. Instead, there are Create and Update triggers that are associated with the **Save** button in the user interface. The list of triggers is provided in the Rule Triggers table.

- Whether the specified trigger is available for the indicated rule type.

  This needs to be taken into consideration for the rule types other than custom or scheduled actions. For example, you cannot create approval rules for the Update trigger or the Check-In trigger in documents.

- Whether the rule may be created through the user interface.

  For example, just because a rule is of type custom action, it does not necessarily mean that its qualifier must be automated as well. Simple qualifiers, such as checking a value in a field or a user's group membership, may often be created through the user interface.

- Whether any automated actions and qualifiers may be reused. Indicate which ones and for which rules.

  For example, several rules may have to perform the same action but under different conditions, or vice versa, perform different actions under the same condition. Always keep in mind the

concept of separating qualifiers and actions in a rule that allows you to mix-and-match files to minimize duplication of effort and to maximize the flexibility of the rule development process.

- Whether there are any complementary rules, rules with the same qualifier and action but a different trigger. Indicate which ones.

  For example, two rules may need to be created to enforce the workflow process when the **Save** button is clicked--one rule with the Create trigger and the other with the Update trigger.

- Whether any additional or complementary rules are needed to enforce the workflow process. Indicate which ones.

  For example, you may need to break a more complicated rule into several rules, or a rule with the Update trigger may be required to complement the specified rule with the Create trigger.

- Whether there is any missing design information that you would need in the rule.

  For example, categories' full tree positions may not be provided for custom fields, or lookup items' full tree positions for List fields or sub-objects, and so on.

Rule Component Separation

Depending on the rule type and business requirements, either component may be defined through the user interface or written in a separate Java or JavaScript file (for details on which rule types may have automated qualifiers and actions, see the Rule Types table). In either case, strive to maintain the separation between the two components, especially when writing automated qualifiers and actions.

Writing each rule component in a separate file allows you to mix and match qualifiers with actions in different rules by reusing files. For example, the same component files may be used for two rules, one with the Create trigger and the other with the Update trigger. This division between the qualifier and the action is also useful in rules when one of the components is defined through the user interface and the other is created through the API.

There is nothing to stop you from placing all of your rule code in a condition to enable multi-stage interactions between condition requirements and actions taken, and having your action file do nothing. However, if you want to maintain an object-unspecific rule set, and be able to mix-and-match qualifiers and actions at will, it is advisable to keep your qualifiers and your actions granular and separate.

Legacy Rules

Rules for TeamConnect versions preceding version 1.6 were written as a single file, with no division between qualifiers and actions. These rules are now called legacy rules, and have a different method of inclusion into TeamConnect from rules for the TeamConnect 2.x architecture. Inclusion methods for legacy rules will be covered in the future versions of this document.

1.1.13 Conditional Expressions

Conditions contain one or more expressions that point to records or other parts of TeamConnect. Conditions also contain a setting for the combination of expressions. Depending on this setting, the condition is true or false.

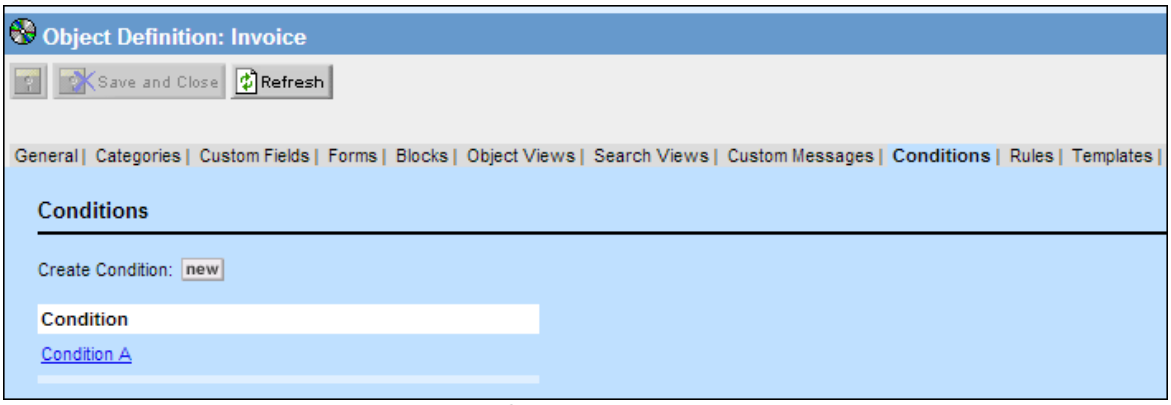
Use Case for Conditions

Use a condition to base the stops in a route on these expressions.

- You can create a route so that different users can approve invoices, depending on the total amount on the invoice. For example, you can have a different user approve each of the following situations:
  - Invoices amounts less than or equal to $1,000.
  - Invoices amounts greater than $1,000 and less than or equal to $5,000
  - Invoices amounts greater than $5,000.
- You can create a route so that if users try to delete a record that requires approval for deletion, different groups can approve the deletion based on different fields in the record.

Finding Conditions

Each object definition has its own set of conditions. To find conditions for an object definition, click the **Conditions** tab from the object definition.



Conditions Tab

1.1.13.1 Creating a Condition

You create conditions from the **Conditions** tab of a object definition.

To create a condition

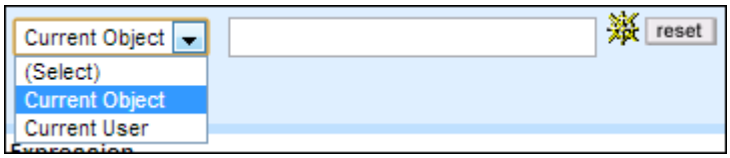
1. From the **Conditions** tab for an object definition, click **New**.
2. Enter information about the condition. See the General Tab of a Condition table for a description of each field.
3. Click **Save**.

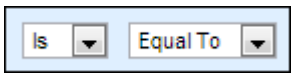
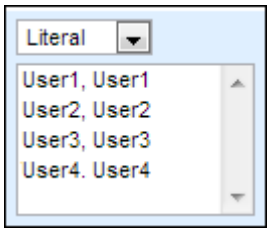
Note: *Click the* ***Delete*** *button of an open condition to delete it.*

To create a copy of a condition

1. From the condition, click **Create a Copy**.
2. Change the **Name** and the **Unique Key** of the condition.
3. Click **Save**.

General Tab of a Condition

Field or Column	Description
Name	The name of the condition. The name you enter is also the name that appears in any drop-down list of conditions.
Unique Key	The unique name of the condition.
Description	Additional information about the condition.
Expression Type	• **Automated Qualifier**—Select this option to add a Java class or JavaScript file as an expression for the condition. • **User Expression**—Select this option to add an expression with TeamConnect fields to the condition.
Expression (when **Automated Qualifier** is selected)	A drop-down list with Java class and JavaScript files that you can select for this condition. Click the **Automated Qualifiers Folder** link to open the folder with Java classes and JavaScript files. For examples of when to create automated qualifier, see When to Create Automated Qualifiers.
Expression (when **User Expression** is selected)	The following fields appear: • **Left Argument**—Select the path you want to create for the condition. ○ Select **Current Object** if you want to use a field in TeamConnect as the qualifier, for example **Current Phase**. ○ Select **Current User** if you want to compare the identity of each user with a field that identifies a user. Click the icon to open the Object Navigator and select the path to the field.  **Left Argument**

	• **Operator**—Select the appropriate way to connect the right and left arguments. See the Operator Fields for Creating Expressions table for more information about operators.  **Operator** • **Right Argument**—Select the attribute or literal value that you want to compare to the attribute that you selected for the left argument. See the Rule Qualifier Right Argument Options table for more descriptions of each field.  **Right Argument**
add more	Click this button to add the expression to the condition.
Expressions table	This table lists all the expressions that you have added to the condition. Place a check-mark in the check-box of an expression and click **edit** to edit the expression. Place a check-mark in one or more check-boxes and click **delete** to delete the expression(s).
Condition is true when	Select one of these options to determine which expression in the condition must be met: • **All of the above expressions are met (AND logic)**—Select this option for all expressions to apply to the condition. • **Any of the above expressions are met (OR logic)**—Select this option for one of the expressions to apply to the condition. • **The following expressions of the above conditions is met**—Enter a formula for the combination of expressions. Use the number of the expression to refer to the expression. See Points To Remember for information about how to create this combination. Click **View Qualifier** to display the qualifier logic visually.

1.1.13.1.1 Defining Expressions

To define an expression, you specify the following:

- The left argument
- The operator
- The right argument

These three pieces together form a statement, such as these:

Sample Expressions

Left Argument	Operator	Right Argument
The value selected in the Loss Type list field	is	Fire Damage
The current user	is not	a member of the Outside Counsel group
The Trial Date	is changed	from any date to any date
The Expiration Date field	is populated	[none]

If you add these statements as expressions, the condition evaluates the statements to determine whether the condition is true.

The type of data being compared in the left and right arguments must match. For example, if your left argument identifies a list field, then your operator and right argument options allow you to compare the selection with a selection made in another list field or to a literal list value.

Operator Fields

The operator connects the rights and left arguments. Refer to the following table for a description of each operator field.

Operator Fields for Creating Expressions

Selection	Description	Data types where operator selection is available					
		Text	Number	Date	List	Check-box	Related object
Is	Determines whether the qualifier statement is a positive or negative statement.	x	x	x	x	x	x
Not		x	x	x	x	x	x
Equal To	Evaluates whether the left argument and right argument are the same value.	x	x	x	x	x	x
Populated	Evaluates whether the field identified in the left argument has a value.	x	x	x	x		x

	Do not use this selection to check whether a field that was previously null now has a value. Instead, use **Changed** from **No Value**.						
Changed	Evaluates whether the value for the field identified in the left argument has changed. It compares the old value in the database to the value in the record when rule is triggered. ***Tip:*** *This option is useful for triggering audit rules.*	**x**	**x**	**x**	**x**	**x**	**x**
Begins With	Evaluates whether the text value in the field identified in the left argument begins with, ends with, or contains the text that is identified in the right argument.	**x**					
Ends With		**x**					
Contains		**x**					
Less Than	Evaluates the number value identified in the left argument compared to the number value in the right argument. For example, you may specify that if a dollar amount **Is—Greater Than Or Equal To** 5000, send the action for approval.		**x**				
Less Than Or Equal To			**x**				
Greater Than Or Equal To			**x**				
Greater Than			**x**				

Exists	These options appear when you select a particular category by creating a path such as `.detailList.(Motions)`. Evaluates whether the category has been added to the record.						
Does Not Exist							
Item Added	Evaluates whether a related object (such as involved) or sub-object (such as assignee) was added, deleted, or modified.				x		
Item Deleted					x		
Item Modified					x		
After	Compares the date in the left argument to the date in the right argument. An additional field appears when you select one of the options that requires you to specify a number of days. Enter the value for **X** in this field.			x			
Before				x			
After The Next X Days				x			
Within The Next X Days				x			
Within The Last X Days				x			
Before The Last X Days				x			

Right Argument Fields

The types of right arguments you can add depend on the data type of the left argument.

Rule Qualifier Right Argument Options

Selection	Description	Data types for which selection is available					
		Text	Number	Date	List	Check-box	Custom Object/ Involved
Literal	Select this option when you want to specify a value to compare to the field identified by your left argument. For example, select **Literal** for a qualifier item that checks whether the invoice total is greater than 5000. You must also select **Literal** when you want to check whether a certain item is selected in a List field.	x	x		x	x	x
Attribute	Select this option when you need to compare the values of two fields in TeamConnect. After you select **Attribute**, you must first select **Current Object** or **Current User**, and then use Object Navigator to identify the field to which you want to compare the field identified by your left argument. For example, if you want to compare the Contact of a contact-centric project to the Contact identified in a custom field of type	x	x	x	x	x	x

	Involved, select **Attribute** to identify the custom field.						
Current Object	Select one of these options after you have selected **Attribute** and you need to identify a field in TeamConnect to which you are comparing the field in your left argument. For details about using Object Navigator, see Using Object Navigator.	**x**	**x**	**x**	**x**	**x**	**x**
Current User		**x**	**x**	**x**	**x**	**x**	**x**
Any Value	These options work in conjunction with the **Changed** option to form a statement, such as **Is changed from any value to this value (California**).	**x**	**x**	**x**	**x**	**x**	**x**
No Value		**x**	**x**	**x**	**x**		**x**
This Value		**x**	**x**		**x**		**x**
Today	The date that the rule is being triggered.			**x**			
This Date	Allows you to compare the date in the left argument to a date that is identified in another field.			**x**			
From Now	Allows you to specify a number of days from the date that the rule is being triggered.			**x**			
From This Date	Allows you to specify a number of days from a date that is identified in another field.			**x**			
Yes	These options correspond to a check-					**x**	

No	box being selected or not selected.					**x**	

1.1.13.1.2 Qualifier Logic

The **Designer** user interface is designed for you to complete the condition logic in stages. After listing one or more expressions on the **Qualifier** tab, the following options appear, allowing you to specify when the rule action runs:

- All of the expressions are met.
- Any of the expressions are met.
- The expressions are met according to the logic that you specify.

The "logic" option allows you to specify a formula for how the condition evaluates the expressions. You refer to an expression using the number that is automatically generated on the batch screen for each item. For example, once you create a list of five expressions, you might create the following formula:

```
(1 or 2 or 3) and (4 or 5)
```

Points To Remember

Keep the following points in mind while working with qualifier logic:

- Define all of the expressions that you need in your condition before writing the formula.
- The only words that you may use in your formula are "and" and "or."
- Use parentheses () to logically group expressions. You may also nest parentheses when needed. Make sure to use complete pairs of parentheses.
- If you do not group the expressions where grouping is necessary, grouping is automatically applied by TeamConnect, and you will have to verify that the appropriate logic is achieved. For example:

  ```
  1 or 2 and 3
  ```

  is automatically converted to:

  ```
  1 or (2 and 3)
  ```

  It is best to apply the grouping yourself to ensure that it is correct.
- If necessary, your formula may repeat the same expression. For example, the following formula is valid:

  ```
  (1 and 2 and 3) or (1 and 2 and 4)
  ```

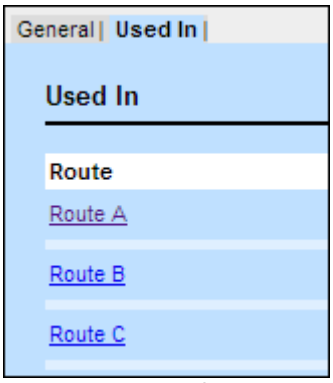
1.1.13.2 Finding Routes that Use a Condition

From an existing condition, you can find out whether one or multiple routes already use that condition.

To view routes that use a condition

1. From the condition, click the **Used In** tab.

   You see a list of routes that use the condition.



Used In Page of a Route

2. Click the link for a route to find out more about how it uses the condition.

1.1.14 Adding Custom Code to TeamConnect

Once you have written and compiled your automated qualifier and/or automated action Java classes or finished writing your JavaScript and XML files, you must include them in TeamConnect. To do this, you need to upload the files to the appropriate folders in your system's Documents area, and define the rules.

The following sections describe how to include automated qualifiers and automated actions in TeamConnect:

- Uploading Rule Component Files
- Defining Rules that Use Custom Code in the User Interface
- Defining Wizard Page Transition Rules
- Defining Wizard Page Actions

For information on defining non-automated qualifiers and actions through the user interface, see Rule Qualifiers and Rule Actions.

1.1.14.1 Uploading Rule Component Files

To include API rule components in TeamConnect, you must upload the corresponding Java, JavaScript, or XML rule component files to the TeamConnect Documents area. There are three major folders in the Documents area for such files:

- **Top Level/System/Custom Rules**—Upload only supporting Java files that are shared between more than one object definition. These files are not available in the **Qualifiers** or **Actions** tab of the Rule screen in the object definitions, nor in the **Page Actions** tab in wizard screens. These files may be only referenced by the files uploaded to the object definitions' **Rules** folder.
- **Top Level/System/Object Definitions/*objectDefinitionName*/Rules**—In each object definition, the **Rules** folder has three subfolders:

- **Automated Actions**—Upload your files for automated actions for custom action rules and wizard page actions here.
- **Automated Qualifiers**—Upload your files automated qualifiers for regular rules and wizard page transition rules here.

The **Automated Actions** and **Automated Qualifiers** folders are the two locations that you will most frequently use to upload your files.

- **Top Level/System/Libraries**—Allows you to deploy JAR files outside of the **TeamConnect.war** file or the **WEB-INF/lib** folder, which has the advantage of not requiring you to restart TeamConnect. This folder may be used for more than rule files.
- **Top Level/System/StartUp**—Allows you to deploy class files that must be executed when TeamConnect starts. This may be useful for installation scripts, to initialize TeamConnect using custom code, and to check consistency in the application for the session. For more details, see Defining Start Up Classes.

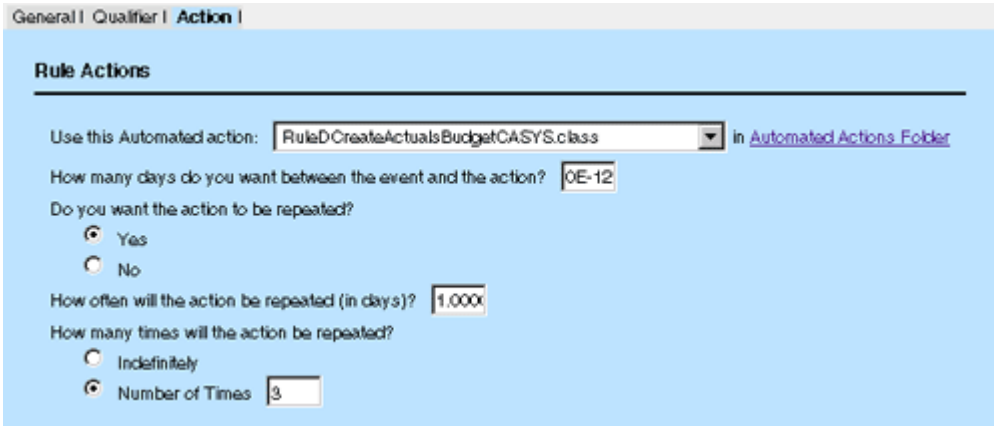
Important: *It is a good idea to limit access to the* ***StartUp*** *folder to prevent the addition of malicious code.*

Ways of Accessing Rule Folders

You may either upload all your files to the respective object definitions before you start defining rules, or you may upload each component file during the definition process of each rule. Aside from your personal preference, this often depends on the way you choose to access the rule folders.

There are two ways of accessing the **Automated Actions** and **Automated Qualifiers** folders where you will typically upload your files:

- From the Documents area by selecting **Documents** in the **Go to** menu in Designer and then navigating to the necessary object definition folder.
- Directly from the **Qualifiers** or **Actions** tabs in the Rule screen by clicking the hyperlink that takes you to the appropriate **Automated Actions** or **Automated Qualifiers** folder of the displayed object definition. For example, see the hyperlink in the following image.



Rule Screen - Actions Tab

This ensures that the files are uploaded to the correct folder, as you do not have to manually navigate to it in the Documents screen.

Uploading Techniques

Depending on the display version of the Documents screen, which may be either HTML or Applet, you may use one of the following ways to upload a file:

- Manually, by using the **Upload File** button on the toolbar.

  You may do this in either version of the screen.

- By dragging and dropping the file(s) from any location on your machine.

  You may do this only in the Applet version of the screen.

For more specific details on the size you can upload a document in TeamConnect, see Managing Documents.

1.1.14.2 Defining Start Up Classes

The **Top Level/System/StartUp** folder allows you to deploy class files that execute when TeamConnect starts. You can use this folder for installation scripts, to initialize TeamConnect using custom code, and to check consistency in the application. Your Java class files can use the TeamConnect's API, other libraries in TeamConnect's **Documents** area, and libraries in other locations.

Important: *You may want to limit access to the* ***StartUp*** *folder to prevent the addition of malicious code.*

TeamConnect finishes its standard initialization before it executes the class files in the **StartUp** folder. Your custom code can affect TeamConnect's initialization after its standard initialization and before TeamConnect becomes available via a web browser.

Note: *Custom code may not be compatible across TeamConnect releases.*

Start Up Settings

Use the **web.xml** file to update settings for the **StartUp** folder.

To enable the execution of custom code in the **StartUp** folder, set the **app.runStartUp** application parameter to **YES** in the **web.xml** file.

To stop the execution of all class files in the **StartUp** folder in the event of any runtime errors or exceptions, set the **app.runStartUp.failOnError** parameter to **YES** in the **web.xml** file.

How TeamConnect Uses Start Up Classes

By default, the system user executes the start up classes. TeamConnect executes classes in the **StartUp** folder under the following conditions:

- The files must be Java classes. TeamConnect ignores non-Java class files and any sub-folders in the **StartUp** folder, unless you reference them from the Java class. For example, you can include JAR files and reference them from class files.

- The Java classes must include implement the `ScheduledCustomAction` interface and the method of `public void action()` to hold the action to run. If you set the system logging level to **Info** or **Debug** and upload Java classes that do not implement the `ScheduledCustomAction` interface, TeamConnect logs the following message and continues to the next java class:

  ClassName does not implement TCStartUpAction or ScheduledCustomAction. Not running ClassName.

- In the alphabetical order of the file names if multiple class files are present in the **StartUp** folder.

1.1.14.3 Defining Rules that Use Custom Code in the User Interface

For all rules that use custom code in some aspect, you must incorporate them into the TeamConnect application after the code has been developed. The following is a reminder of the steps for defining rules in TeamConnect:

1. Setting general rule information, as described in Setting General Rule Information.
2. Defining the necessary action. See:
   - Rule Actions
   - Setting Automated Actions
3. Adding the necessary qualifiers. See:
   - Creating Rule Qualifiers
   - Setting Automated Qualifiers
4. Activating the rule. See Enabling Rules.

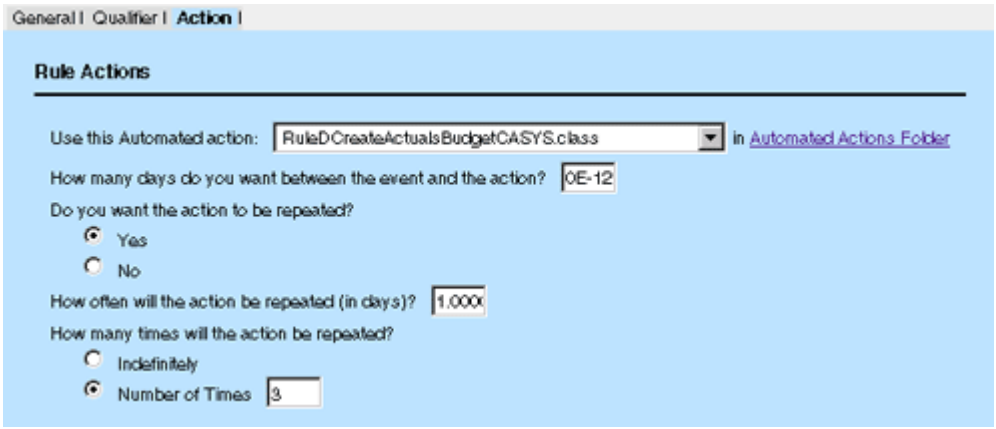
For details on defining Page Transition rules in wizards, see Defining Wizard Page Transition Rules.

To set an automated action

Note: *Automated actions may only be set for rules of type Custom Action.*

1. In the rule screen, click the **Actions** tab.
2. In the **Use this Automated action** drop-down list, select the appropriate automated action file.

   If you have not uploaded your automated action file, click the **Automated Actions Folder** link to proceed directly to the **Automated Actions** folder of the selected object definition. Upload the file, and it will become available for selection in the **Use this Automated Action** drop-down list.
3. If there are any parameters in the action, enter values in the provided fields.
4. Click **Save**.

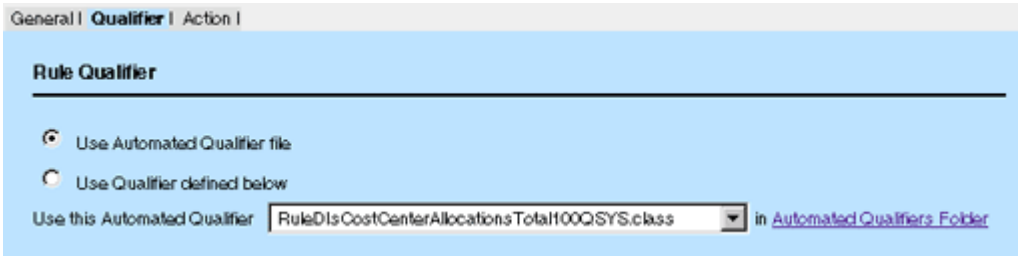


Rule Screen - Actions Tab

To set an automated qualifier

1. In the rule screen, click the **Qualifiers** tab.
2. Click the Use **Automated Qualifier** radio button.
3. In the Use this **Automated Qualifier** drop-down list, select the appropriate automated qualifier file.

   If you have not yet uploaded your automated qualifier file, click the **Automated Qualifiers Folder** link to proceed directly to the **Automated Qualifiers** folder of the selected object definition. Upload the file, and it will become available for selection in the Use this **Automated Qualifier** drop-down list.
4. If there are any parameters in the qualifier, enter values in the provided fields.
5. Click **Save**.



Rule Screen - Qualifier Tab

To enable your rule

1. In the **General** tab of the rule screen, select **This Rule is Active**.
2. If you have defined a user invoked rule, make sure to give the right to that rule to the appropriate users, through the appropriate group accounts.

User Invoked Rules

User invoked rules are, as the name implies, invoked directly by the user. When you define a user invoked rule for an object definition, the rule appears in a drop-down list when a user with the appropriate rights views a record or other user interface associated with that object definition. This operation uses the **More Actions** button in the toolbar, the same button that accesses user invoked actions (see User Invoked Actions).

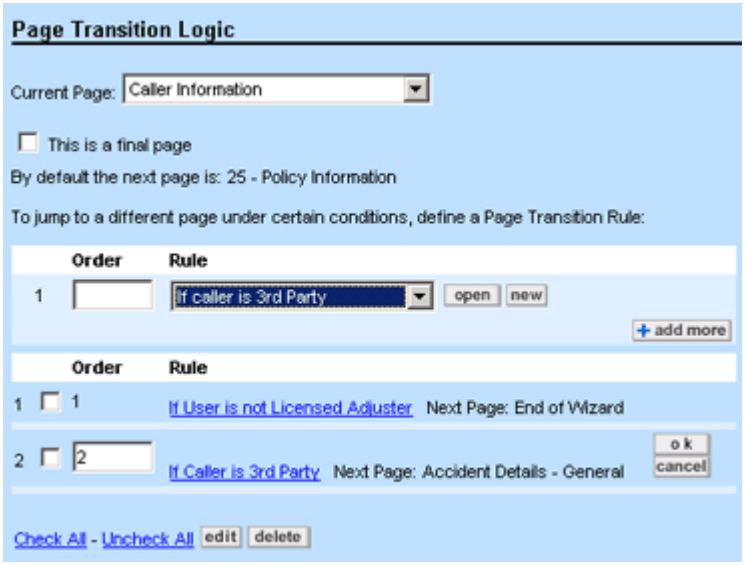
1.1.14.4 Defining Wizard Page Transition Rules

To define a page transition rule in a wizard

1. Click the **Page Transitions** tab in the wizard.
2. Click **new** to open a new rule screen.
3. In the **General** tab, enter a description that clearly identifies the rule.
4. Click the **Action** tab and select the page to which the wizard takes the user if the qualifiers are met and click **Save**.

   Wizard Page Transition rules do not have automated actions, and their actions only indicate to which page in the wizard to go next, when the qualifiers are met.
5. Click the **Qualifiers** tab and make the appropriate selections as explained in Setting Automated Qualifiers.
6. Click **Save and Close**.

   The rule becomes available in the **Rule** drop-down list in the **Page Transitions** tab of the wizard:



Example of Page Transitions Tab in Wizards

   By default, all Wizard Page Transition rules are active. However, they do not take effect until you add them to a wizard page.
7. Select the rule you have defined and enter the order in which it must be triggered when the user clicks next on the wizard page.

8. In the **Current Page** drop-down list, select the page whose transition must trigger the rule and click **add more** or save the wizard.

1.1.14.5 Defining Wizard Page Actions

Currently, you cannot upload automated action files directly from the **Page Actions** tab in wizards. Make sure to upload your files to the **Rules\Automated Actions** folder under the appropriate object definition in the Documents area. For details, see Uploading Rule Component Files.

To define an automated page action in a wizard

1. Click the **Page Actions** tab in the wizard.
2. In the **Current Page** drop-down list, select the page for which you need to define the automated action, and select the **Use Automated Actions** radio button.
3. In the **Use this Automated Action** drop-down list, select the appropriate automated action file.
4. If there are any parameters in the action, enter values in the provided fields.
5. Click **Save**.

1.1.15 Localizing TeamConnect

TeamConnect supports localization - the display of web pages in multiple languages and formatting styles.

The most important part of the localization infrastructure is the *i18n key*. This is a unique value assigned to each element that is subject to localization in the entire application. Examples of such elements are:

- Labels (of tabs, push buttons, fields, etc.)
- Lookup table entries
- Category names
- Messages
- Other static text on the page

Note: *User-entered data is not localized. For example, if an Appointment record is named "North Hills Partners Settlement Conference," that is the text that will be rendered to the Name field of every end user's Appointment web page, regardless of their locale. But the static label next to that field might say "Name" or "Nom" or "Nombre," depending upon the end user's locale.*

The first step in localization is to register all the locales that TeamConnect must support. For details, see Working with Locales. After those locales are registered, you can export your application's design element resources.

All elements that have i18n keys can be exported to a spreadsheet. The sheet contains a column for the i18n key (Message Key), a column for Default Value (the static text, in the system default locale), and an additional column for each other locale that has been registered by the system administrator. To localize the application, enter static text in other languages in those additional columns, then import the edited spreadsheet to the application. For details of these procedures, see Working with Locales.

Design elements that are already unique will have their i18n key generated automatically. For example, a custom field name (when combined with its object definition's unique code and its category) is already guaranteed to be unique. But some design element names are not required to be unique - for example, a report name. In cases like these, the solution developer must assign a value to a Unique Key field in that design element. The Unique Key will be used as part of the automatically generated i18n key. Unique Key fields are found in several places in Designer, as well as in report design pages. Design elements that have empty Unique Key fields will not be exported to the localization spreadsheet.

1.1.15.1 How are i18n Keys Generated?

For "out of the box" static text that is present in TeamConnect before any customization, all i18n keys are already present. Those keys are permanent and are not editable by you. These elements of static text become rows in the localization spreadsheet. Shown below are examples of spreadsheet entries to show the patterns used in generating such i18n keys.

i18n key and default text examples

Design element	Example i18n key and default text	Notes
Static text	account.accountProjectType Posting Project to this Account	
Static text	accountView.title Account - {0}	This message accepts a parameter - the name of the account. When localizing it is important to preserve parameter tokens.
Block	objdef.ACCT.block.Account_Children Child Accounts	
Button text	button.saveAndPreview Save & Preview	
Enumeration	enum.docucontent.26 Rich Text Format	Enumerations can be used to populate fixed drop-down lists, in this case for choosing the file type of a document.

Enumeration	enum.lineitem.adjustmenttarget.netamount Net Amount	
Error message	error.screendesign.tagUsage Could not parse screen file due to improper use of a tag in file. Please check logs.	
Error message	common.typeMismatch Value "{0}" entered for criterion "{1}" is invalid.	Parameter tokens must be preserved when localizing.
Search view (criterion)	objdef.ACCT.searchview.DefaultTemplate.criterion.Vendorr290c1 Vendor	Tokens like "r290c1" were used by prior versions of TeamConnect to control the positioning of a criterion on the filter page.
Search view (result column)	objdef.ACCT.searchview.DefaultTemplate.resultscolumn.Balance Balance	
System lookup table entry	table.system.CURR.item.CAD Canada, Dollars	
Custom field	objdef.CONT.customfield.Eligible Eligibility	"Eligible" is field name. "Eligibility" is field label.

1.1.15.1.1 Features that Require Your Attention

The preceding sections described TeamConnect features whose i18n keys are generated automatically. The sections below describe features where you can influence the generation of i18n keys for localization.

Note: *Many of the features below use* ***Unique Key*** *fields. Characters entered in those fields must be ASCII letters and numbers.*

1.1.15.1.2 Reports

In order to localize report properties such as data series names, column headers, etc., there must exist an i18n key for each property. Since you can design brand-new reports that didn't exist in the out-of-the-box design, you need a way to assign i18n keys to the new reports' properties.

This need is addressed by the **Unique Key** fields that are found in the various report design pages. If you do not plan to localize a report, you may leave these fields empty. But if you need to localize the report, each of these fields in the report must have a value.

For details about localizing reports, see Localizing Reports.

1.1.15.1.3 Custom Messages

Custom messages are created using the **Custom Messages** page within **System Settings** in Designer (for global messages) or using the **Custom Messages** page within an object definition (for object-specific messages). They consist of an i18n key and default text. The i18n key takes two forms:

- **custom.common.someUniqueKey—**for global messages, where "someUniqueKey" is the only portion of the key that is actually entered by the solution developer.
- **custom.INVC.someUniqueKey—**for object-specific messages (object INVC, in this example), where "someUniqueKey" is the only portion of the key that is actually entered by the solution developer.

Custom messages are exported to the localization spreadsheet and you can supply alternate locales' text to the messages in that spreadsheet.

Custom messages can be used in custom actions to provide localized error messages.

Localized text for custom messages is available through the API methods: `localize()` and `localizeNumber()`.

1.1.15.1.4 Custom Blocks

The static text of custom blocks can be localized during design by referencing custom messages rather than typing literal text.

The Screen Designer tool allows you to reference custom messages by key. If you are editing block XML files manually, you can use tags **tc:message** and **tc:messageParam** to achieve the same effect.

Text that is added to a block by entering literal text in the Screen Designer, or by using the **out** tag with literal text, will not be localized. It will always by rendered as it was originally entered.

1.1.15.1.5 Portal Panes

The Designer page for **Portal Pane Settings** provides the opportunity to enter **Unique Key** values for the pane and its contents, which are converted to full i18n keys during export to the localization spreadsheet.

1.1.15.1.6 Rules

Each rule has a **Unique Key** field that is converted to a full i18n key during export to the localization spreadsheet. Unlike some other design elements, the name of a user-invoked rule can be localized so that it appears in the user's locale in the **More Actions** drop-down list at runtime. There are rows in the spreadsheet related to security rules' message text and validation rules' message text, so those can be localized as well.

1.1.15.1.7 Wizards

Wizard design pages have **Unique Key** fields in several places, covering the wizard itself plus its text elements. Unique Key values are converted to full i18n keys during export to the localization spreadsheet.

1.1.15.1.8 Saved Searches

A saved search is one end user's copy of a search view, edited for the needs of that user and saved under a name that is assigned by the user. Saved searches do not have i18n keys, so there is no action to be taken by the solution developer with regard to localization. However, there is some localization behavior that should be understood. Under one uncommon and specific set of circumstances, a saved search can behave slightly differently from other features, with respect to localization.

The labels of criteria and result fields are always obtained, when possible, from the original search view that underlies the saved search. However, since the end user can remove results and criteria, and add brand-new results and criteria, there can be elements in the saved search that are not found in the original search view. In this case, one of two things will happen:

- If the element is a custom field, its label will be obtained from the object definition.
- If the element is not a custom field, the label used will be the one that was in existence at the time the search was saved. Even if that end user is now using a different locale, the label in the saved search will reflect the locale at the time the search was saved.

1.1.15.2 Upgrade Considerations for Localizing

If you upgrade TeamConnect from a version older than version 3.4, your application will be enabled for localization but, at first, only text in the system default locale (English-U.S.) will be available. The installer will take the following actions automatically:

- **System default locale**—Set to English (US).
- The **Region** block of User Preferences previously contained separate fields for country and language. That has been replaced by a **Locale** block containing a single drop-down list in which "English (United States)" is selected by default for all end users.
- Wizard list parameters that use comma delimited strings will still be supported for backward compatibility. But the new model for parameters of type list requires custom lookup tables, and all future editing of such parameters can only be done with custom lookup tables.

For localization to work properly, your database requires additional configuration beyond the basic requirements for TeamConnect. For details, see Database Requirements.

1.1.15.2.1 Custom Blocks

Significant work is required to make custom blocks ready for localization. Custom blocks are defined by XML files which, in previous versions of TeamConnect, will contain literal text such as:

```
<br/>
    * No E-Billing Role Available
<br/>
```

To localize such blocks, all occurrences of literal text must be converted to custom messages such as:

```
<br/>
    <tc:message
    key="custom.CSM$.screen.xml.eBillingRoles.noEbillingRoleAvailable" />
<br/>
```

...where the referenced message key should have been already added via the Custom Messages facility in the object definition for CSM$. Such custom messages are visible in file **custommessages.xml**, after a design snapshot export, in the following format:

```
<CustomMessage
messageKey="custom.CSM$.screen.xml.eBillingRoles.noEbillingRoleAvailable"
messageDefaultText="* No E-Billing Role Available"/>
```

1.1.15.2.1.1 Parameters

Text that used dynamic values will need to be rewritten to use the tc:messageParam tag. For example, a message that said:

```
Share documents to <tc:out value="${cjb.vendorName}" /> in Collaborati.
```

Becomes:

```
<tc:message
key="custom.VEN$.screen.xml.vendorDocumentSharing.shareDocumentsToVendor">
    <tc:messageParam value="${cjb.vendorName}"/>
</tc:message>
```

...and you create a corresponding custom message with key "custom.VEN$.screen.xml.vendorDocumentSharing.shareDocumentsToVendor" and text "Share documents to {0} in Collaborati."

1.1.15.2.1.2 Date Formatting

The example below shows date formatting and text localization. A custom message is supplied to the `localize()` method.

```
public String getLastPublishedOnString() {
    Date lastPublished = new Date();
    String localizedDate =
    platform.getUtilityService().formatDateByUserSetting(lastPublished);
    return
    platform.getInternationalizationService().localize("customKey.publishedDateMess
    age", localizedDate);
}
```

1.1.15.2.1.3 Best Practices

Avoid using custom java code to simply append dynamic values in a message.

If the label of a field was previously hard-coded in a custom screen, use `<tc:message>` instead of `<tc:label>` to keep exactly the same text formatting.

Avoid making new custom messages for field labels if an existing label will do the job. For example, `<tc:label label="Company Tax ID" />` could be replaced by the object definition field label `<tc:label category="VEN$" name="companyTaxID" />`, presuming that the field label from the object definition already says "Company Tax ID". The label from the object definition will have been localized already, so this approach avoids double work.

1.1.16 Migrating Custom Designs

The typical life cycle of any new TeamConnect design is:

- Design/Development phase
- Testing phase
- Production phase

The following design tools are provided for migrating custom designs from a development TeamConnect instance to a QA or production TeamConnect instance:

- **Design Snapshot Tool**—From your development TeamConnect instance, captures your latest design snapshot, provides a summary of archived design snapshots, generates a design update file (used to update a TeamConnect instance at custom design version x to design version y). You may also revert your current design to a previously saved snapshot using this tool.
- **Design Import Tool**—From your QA or production TeamConnect instance, applies a design update file to upgrade the instance from design version X to design version Y.
- **Configuration Transfer Utility**—Allows users to transfer custom designs and configurations to a new production version of TeamConnect.

Caution: *It is neither recommended nor supported for a user to make separate changes to both the development and QA instance designs, and then use the Design Snapshot Tool and Design Import Tool to merge the respective designs.*

For descriptions on assigning rights to work with the Design Snapshot Tool and Design Import Tool, see Group Tool Rights.

1.1.16.1 Requirements

For the design import process, it is required that both the source (development) and destination (QA or production) TeamConnect instances are at the same version.

1.1.16.2 Working with the Design Snapshot Tool

Use the Design Snapshot Tool from your development TeamConnect instance to:

- Capture design snapshots
- Create design upgrade files
- Revert to previous snapshots

For information about setting rights to use the Design Snapshot Tool, see Group Tool Rights.

Design Snapshot Tool

1.1.16.2.1 Creating a Design Snapshot

After you make your custom design changes on the development TeamConnect instance, you can use the Design Snapshot Tool to capture a snapshot of the current design. It is recommended that you take regular snapshots of your design to back up changes.

To create a design snapshot

1. From the Designer, **Tools** drop-down list, select **Design Snapshot Tool**.
2. From the **Create Snapshot** section, type notes about design changes implemented. (Be detailed in your description, because you may need good notes at a later time if you perform a Revert operation as described in Reverting to a Previous Snapshot.)
3. Click **Create Snapshot**.
4. Click **OK** in the confirmation dialog.

   Afterward, the new design snapshot is added to the **Snapshot Files** section below.
5. If the export does not complete, click the **View Log** link. For more information, see the Troubleshooting and Warnings section.

1.1.16.2.2 Creating a Design Upgrade File

After you've captured one or more design snapshots, you can create a design upgrade file using the Design Snapshot Tool from the development TeamConnect instance.

Note: *The first design snapshot file, Design0.zip, represents the out-of-box TeamConnect design and is provided for you.*

To create a design upgrade file

1. From the Designer, **Tools** drop-down list, select **Design Snapshot Tool**.
2. From the **Design Versions** section, select the current design version of the target TeamConnect instance (QA or production).
3. Also select the latest design version that you would like to upgrade the target instance to.

   For example, if you have captured two snapshots on the development TeamConnect instance, and would like to upgrade your out-of-box QA TeamConnect instance to the second snapshot, select Design0.zip and Design2.zip (where Design0.zip is the default snapshot provided for you).
4. Click **Generate Import File**.
5. From the confirmation pop-up window, click **Save**.

   If the file generation does not complete, click the **View Log** link. For more information, see Troubleshooting.

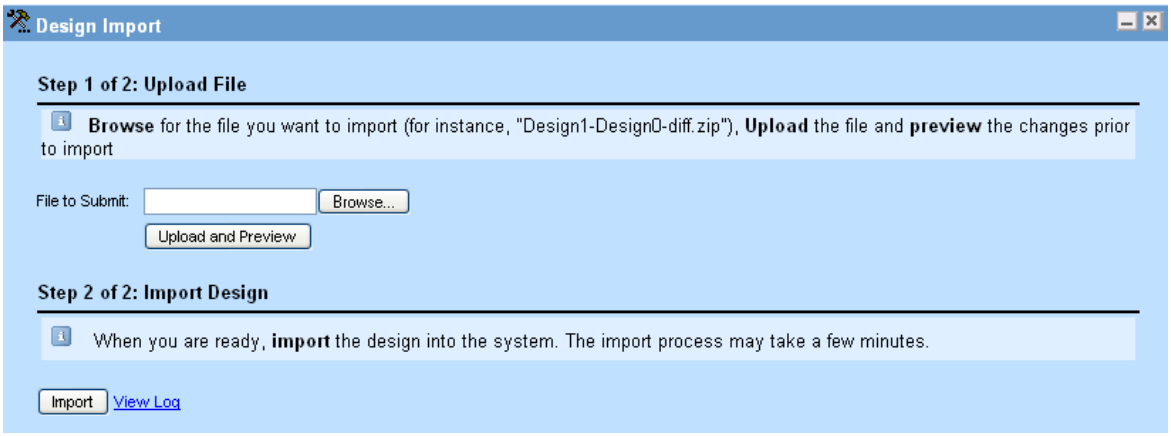
1.1.16.2.3 Reverting to a Previous Snapshot

From the Snapshot page, you can revert your design to any existing, previously captured snapshot. To do so, click the button **Revert to this version** next to the desired snapshot. TeamConnect will then capture a snapshot of your current design, generate a diff file between your current design and the desired snapshot, and import that diff file to your design.

1.1.16.3 Working with the Design Import Tool

Use the Design Import Tool from your QA or production TeamConnect instance to upgrade the current design to a newer design version that was created on your development TeamConnect instance. It is recommended to perform a design upgrade to a production TeamConnect instance during a pre-scheduled period of application down-time.

For information about setting rights to use the Design Import Tool, see Group Tool Rights.



Design Import Tool

1.1.16.3.1 Performing a Design Upgrade

After you've generated a design upgrade file, you can use the Design Import Tool from a QA or production TeamConnect instance to upgrade to a target design version.

Before you begin, back up the destination TeamConnect database.

Important: *Since both a database backup and a design import could require significant resources, Mitratech recommends that you perform design import during off-peak hours on production application servers.*

Note: *If the QA or production TeamConnect instance that you are upgrading is not at the same base design version supported in the design upgrade file, then the import will not complete. For example, if the QA TeamConnect instance is at design version 3 and the design upgrade file updates the design from version 4 to version 5, then the import will not complete.*

To perform a design upgrade

1. From the Designer application, **Tools** drop-down list, select **Design Import Tool**.
2. From the **Step 1 of 2: Upload File** section, click **Browse**. Navigate to the location of the design upgrade file.
3. Click **Upload and Preview**.

   A pop-up window appears displaying a summary of design changes that results from the design import.
4. Click **Done** after you finish reviewing the summary to close the window.
5. From the **Step 2 of 2: Import Design** section, perform the import. If you wish to take a snapshot of the current design just before performing the import, click the relevant checkbox. Click **Import**. Click **OK** in the confirmation pop-up menu.
6. DMT checks the existing design to see if any incompatibilities exist with the design that is about to be imported. If any are found, warning messages are displayed. In some cases, you

are given the option to ignore the warning and continue with the import. For more details about possible warning messages, see Warnings.

Depending on the extent of design changes, the import may take a few minutes or longer. The status icon at the upper right turns while the import is in progress.

7. After the import completes, a success message appears. If the import does not complete, and an error displays, click the **View Log** link. For more information, see General Troubleshooting Steps.

Even when your import completes with a success message, you should scan the log file for warning messages. Some items, such as object views for embedded objects, are ignored during import. A warning will be generated when an item is ignored. You should be aware of such situations.

1.1.16.4 Working with Configuration Transfer Utility

Like the older version of the Design Import Tool, the new Configuration Transfer Utility (CTU) allows users to transfer the current design that was created on your development TeamConnect instance to a different design version in your Production instance.

Use the CTU to make small incremental changes (e.g. add a rule for a custom object), medium changes (create a set of workflows), and large changes (provision a new server).

The process for using the new CTU is slightly different and changes have been made on how artifacts and design dependencies are handled.

For instance, in an example scenario a user has added a rule and then later made modifications three separate times. All entries are captured automatically by the tool, and in order to export the change the user needs to include all four entries (the initial rule creation and subsequent changes; i.e. the dependencies) in the export package. Packages can be in one export file or split up, but the import must follow the sequence that creates the object first, then the rules on which they depend. This differs from the old Design Import Tool in that, previously, the package would have only one entry for that rule creation.

Note: *The previous Design Import Tool could sometimes be used as a design backup, in that if an upgrade failed users could revert back to the old design, removing any partial changes. CTU does not have this capability.*

The Old Way: **Design Migration Tool**	**The New Way:** **Configuration Transfer Utility**
1. Make changes in Development environment	1. Make changes in Development environment
2. Export entire configuration as a design snapshot	2. Filter and select only the changes you want
3. Edit XML files in the snapshot to select the changes	3. Export the changes to a .zip package

The Old Way: Design Migration Tool	The New Way: Configuration Transfer Utility
4. Import the edited snapshot	4. Import the package by uploading it
5. If import fails, repeat steps	

Click the links below for instructions on using CTU:

- Export Design Changes
- Import Design Changes
- Troubleshooting and Best Practices
- Artifacts Captured for Custom Designs

Important Note: *In order to use the CTU, the user's Group must have Contact, Document, and View Rights for* ***Admin Settings****. If the user does not have Contact Rights they cannot see the Changed By nor the Create By values; if the user does not have Document Rights they are unable to view Export Packages.*

1.1.16.4.1 Export Design Changes

As a TeamConnect administrator, you can select appropriate changes that were made in **TeamConnect Setup** or **Admin Settings** to be imported to your next environment. This provides the flexibility to migrate designs in parts depending on the demands of your project, as well as allow for precise scrutiny over the changes to be made.

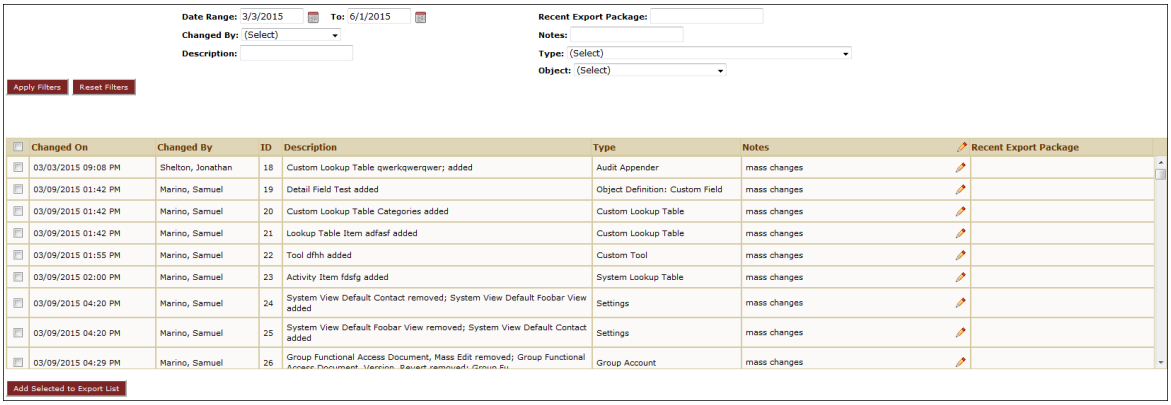
Accessing the Export Design Changes page:

From your TeamConnect homepage, click or hover your mouse pointer over the **All** tab. Select **Export Design Changes**, located under the **Tools** section.

The data found in the top Design Changes table records changes made in TeamConnect Setup (ex: removing, changing, or adding look-up tables). Changes are listed by the date and time the change was made (starting with the most recent), who made the change, change ID number, description of the change, the category (Type) of change, and any notes appended. Each design change is associated with a design category, shown in the **Change Type** column (e.g. Audit Appender, Custom Tool, Global Navigation, Group Account, etc).

If a change has previously been exported, the file name of that package appears in the **Recent Export Package** column. Identifying previously exported packages streamlines the process of making corrections or additions to a group of changes, and focuses attention to packages that have not yet been exported.

If a change has been previously imported, an [IMPORTED] indicator will appear in the **Description** column.



Design Changes Table and Filter

Use the filter tool above the table to show only changes by date range, individual who made the changes, description of changes, note keywords, type, or object (e.g. Appointment, Line Item, Task, etc.). Click **Apply Filters** to sort the list, or **Reset Filters** to list all design changes. The **Changed By** filter layout depends on settings defined in **Admin Settings | User Interface | User Selection**.

If a design change has been made under Admin Settings, the **Change Type** column will display "Admin Settings".

Control the amount of design changes displayed in the table by by selecting an option from the Design Changes per Page drop-down menu, located at the bottom right corner of the table. Navigate through the design change pages by clicking the numbers under the table. ***Warning:*** *Moving from page to page resets the checkboxes you selected. Click* ***Add Selected to Export List*** *before you move on to the next page.*

To include a note or details for an individual export change, click the pencil icon under the **Notes** column, type into the text box and click the check mark to confirm or the red "x" to cancel. Notes are transferred when importing these design changes.

To include a note or details for multiple design changes in the table, check the boxes of the items for which you would like to include a note, then click the pencil icon located inside the **Notes** column. A pop-up window appears where you can type in your notes or instructions. Click **Apply** to confirm or **Cancel** to exit without saving. ***Note:*** *If none of the checkboxes are marked, clicking the bulk edit icon has no effect.*

Edit Notes

To create an export package, use the following steps:

1. Click any number of checkboxes from the Design Changes table.
2. After selecting the desired design changes, click **Add Selected to Export List**, located at the bottom of the table. The page refreshes to reflect your selections in the Export List Summary table.

   Note: *TeamConnect issues a warning if you export changes from multiple versions of TeamConnect, and you can see what versions are included in the* ***TCE Version*** *column. You also receive a warning if there are non-consecutive changes added to your export list. A non-consecutive change is considered a "skip" in the numeric sequence of ID numbers (i.e.*

exporting 1, 2, 4 would issue a warning because ID number 3 has been omitted). The warnings are an opportunity to correct unintentional changes, and do not prevent the user from dismissing the warning and continuing with the export.

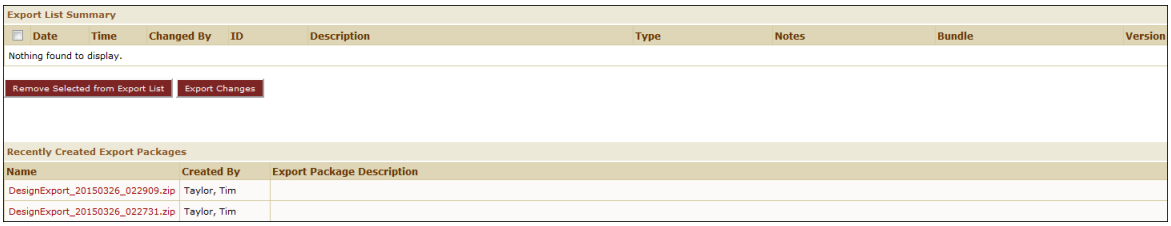
The items in the Export List Summary table are listed by change ID numbers in ascending order and match the ID numbers in the table from which you initially selected the items.

3. Click **Export Changes**. To remove a change from this selection, click the corresponding checkbox and click **Remove Selected from Export List**. You do not need to select the checkboxes in order to export the changes.

   When exported, each change includes a list of host names of the servers on which the change was made and any servers on which it was previously imported. It also indicates the user who originally performed the changes. When a change that has be imported is re-exported, the user identified as the one making the change will be displayed as the one who imported it.

Export List Summary

Upon clicking **Export Changes**, a text field appears in a pop-up window. Using the Export List Summary field, add notes about individual changes or a description of the export. Click **Finish Export** to confirm.



Export Changes

Package files can be downloaded from the Recently Created Export Packages table (in the bottom table, shown above) by clicking the hyperlink under the **Name** column.

1.1.16.4.1.1 View Older Packages

Because the **Export Design Changes** page only displays the ten (10) most recently created export packages, you can click the **View older packages here** hyperlink at the bottom of the screen to view all export packages created by any user in the TeamConnect application.

Recently Created Export Packages

Name	Created By	Export Package Description
DesignExport_20150326_022909.zip	Taylor, Tim	
DesignExport_20150326_022731.zip	Taylor, Tim	
DesignExport_20150323_080219.zip	Shelton, Jonathan	
DesignExport_20150323_080210.zip	Shelton, Jonathan	
DesignExport_20150323_075733.zip	Shelton, Jonathan	
DesignExport_20150323_075731.zip	Shelton, Jonathan	
DesignExport_20150323_075631.zip	Shelton, Jonathan	
DesignExport_20150323_075625.zip	Shelton, Jonathan	
DesignExport_20150323_075516.zip	Shelton, Jonathan	
DesignExport_20150323_070917.zip	Shelton, Jonathan	

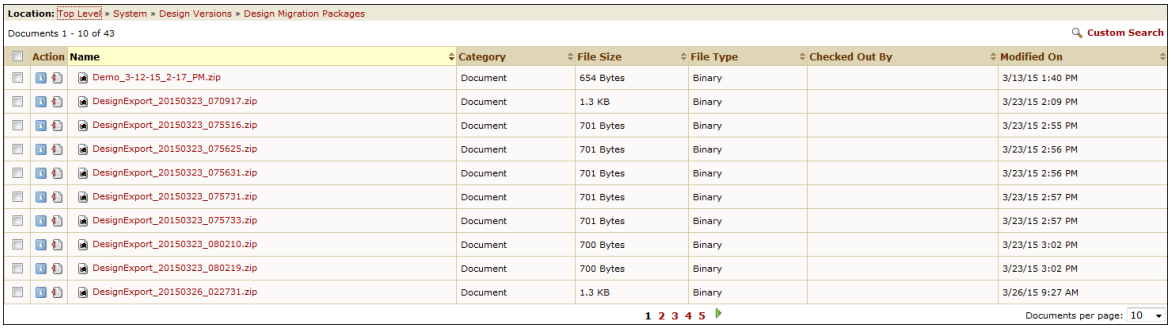
View older packages here

View Older Packages Here Hyperlink

The Design Migration Packages table records the package name, category, file size and type, the name of the individual who has checked out the package (if applicable) and most recent date and time the package was modified.

Additionally, under the **Action** column, users can view package properties and/or check out the package.

- —View package properties
- —Check out package



Location: Top Level » System » Design Versions » Design Migration Packages

Documents 1 - 10 of 43 — Custom Search

Action	Name	Category	File Size	File Type	Checked Out By	Modified On
	Demo_3-12-15_2-17_PM.zip	Document	654 Bytes	Binary		3/13/15 1:40 PM
	DesignExport_20150323_070917.zip	Document	1.3 KB	Binary		3/23/15 2:09 PM
	DesignExport_20150323_075516.zip	Document	701 Bytes	Binary		3/23/15 2:55 PM
	DesignExport_20150323_075625.zip	Document	701 Bytes	Binary		3/23/15 2:56 PM
	DesignExport_20150323_075631.zip	Document	701 Bytes	Binary		3/23/15 2:56 PM
	DesignExport_20150323_075731.zip	Document	701 Bytes	Binary		3/23/15 2:57 PM
	DesignExport_20150323_075733.zip	Document	701 Bytes	Binary		3/23/15 2:57 PM
	DesignExport_20150323_080210.zip	Document	700 Bytes	Binary		3/23/15 3:02 PM
	DesignExport_20150323_080219.zip	Document	700 Bytes	Binary		3/23/15 3:02 PM
	DesignExport_20150326_022731.zip	Document	1.3 KB	Binary		3/26/15 9:27 AM

1 2 3 4 5 — Documents per page: 10

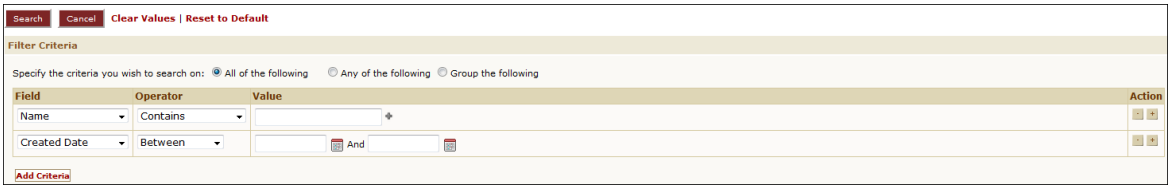
Design Migration Packages Table

To filter the table layout, use the arrows in each column header (⇕) to sort data alphabetically, chronologically, or by size. Use the Documents per page drop-down menu at the bottom right of the table to control how many documents you see per page.

Open a package with Windows Explorer by clicking the hyperlink under the **Name** column. In the new pop-up window, select the option to open or save the file to your computer, then click **OK**.

For additional filter criteria, click **Custom Search** at the top right of the table.

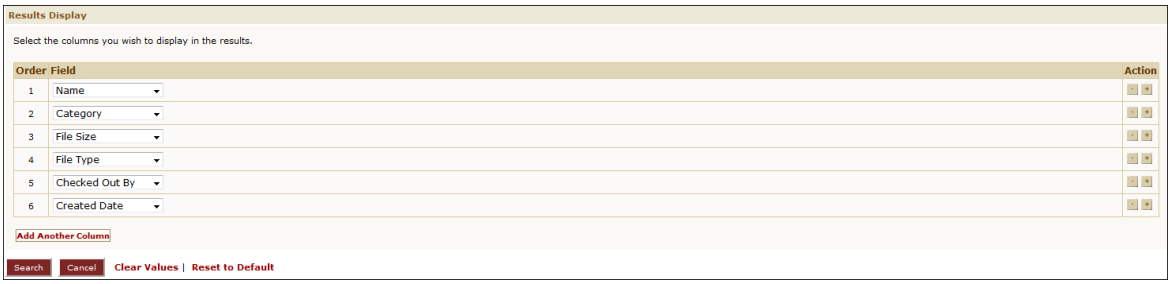
By clicking the **Custom Search** hyperlink in the **View Older Packages** page within **Export Design Changes**, you can further filter the results for design migration packages.



The top portion of the **Custom Search** page allows you to specify the criteria on which you wish to search by value of a particular field (e.g. package name, category, file type), or date.

- Using the radio buttons above the Filter Criteria table, determine whether you want to include all of the following values, any of the values, or group the values.
- Select a field and operator from the drop-down lists, then type in the value for which you wish to search. For the dates section, click on the icon to select a date.
- Under the **Action** column, use the plus or minus buttons to add or erase additional search fields, or click the **Add Criteria** button at the bottom of the table for the same effect as the plus button.
- Click **Clear Values** at the bottom of the page to remove text under the **Value** column, or click **Reset to Default** to remove all values and added search fields.

After filling in all search criteria, click the **Search** button to apply the filters. Click **Cancel** to return to the previous page.



The bottom portion of the **Custom Search** page allows you to select the columns you wish to display in the Design Migration Packages table.

- Under the **Order Field** column, make a selection from the drop-down lists. If you would like to remove one of the columns, leave the drop-down list on the "(Select)" option, or click the minus button under the **Action** column.
- Add a column by clicking the plus button under the **Action** column, or click the **Add Another Column** hyperlink at the bottom of the table.

  Note: *Unless you choose a field from the drop-down list, the added column does not appear in the Design Migration Packages table.*

- Click **Clear Values** to remove text under the **Value** column, or click **Reset to Default** to reset fields and remove added columns.

After choosing your display criteria, click the **Search** button to apply the filters. Click **Cancel** to return to the previous page.

Viewing Your Search Results

Once you have entered your search criteria and selected **Search**, you can view the modified Design Migration Packages table.

In the new screen, you have additional hyperlinks above the table.

Additional Options

- **Custom Search—**Return to the **Custom Search** page with cleared values.
- **Save Search**—Save your current custom search criteria.
- **Modify Search**—Return to the **Custom Search** page with your current values in place.

1.1.16.4.1.2 System Settings Excluded From Export

As part of the **Configuration Transfer Utility** tool, certain changes made in **Admin Settings** are captured to be imported into new environments. A number of system settings are excluded from the design migration tool, and these items are listed below:

Admin Settings Tab	Excluded Setting
General	Business Objects
	TeamConnect URL
Region	Default Locale
	Default Currency
Calendar	All Settings
Email	Outgoing Mail Server Settings
	Incoming Mail Server Settings
	TeamConnect IMAP Server Settings
Maintenance	Maintenance Notification
	Shutdown Alert
Connections	Proxy Settings

Admin Settings Tab	Excluded Setting
	Collaboration Portal
History	All Settings
About	Instance Information
	Available Updates
	Support Options

Click **Update** to add changed system settings. Users must update first before exporting, or the changes cannot be imported.

1.1.16.4.2 Import Design Changes

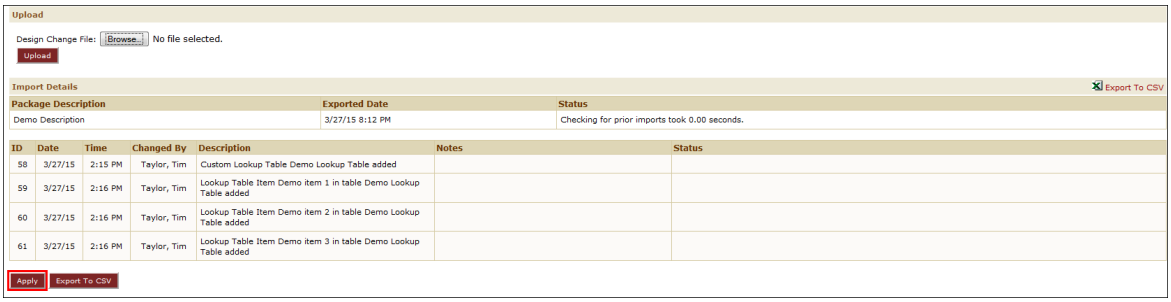
As a TeamConnect Administrator, you can create design migration changes by importing a package not only made to the target environment, but also logged with Data Migration Tools. This ability aids in pinpointing changes that have already been imported, and identifying changes that still need to be made. Additionally, you can create an audit trail for changes that are being made to an environment where they potentially should not be. The import operation runs on foreground processing, as opposed to background processing, so the transaction can run reliably and unattended.

Accessing the Import Design Changes page:

From your TeamConnect homepage, click the **Tools** button in the upper right corner of TeamConnect. Select **Import Design Changes**.

At the top of the **Import Design Changes** page, select **Browse** to import your package. Once you have selected the appropriate file from your computer, click **Open**, then click **Upload**.

Your imported package appears in the Import Details table. If any notes were made in the **Export Design Changes** page, they appear in the **Notes** column. ***Note:*** *Uploading an incompatible file does not display results in the table.*

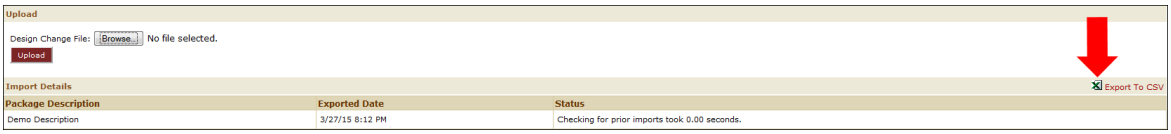


Top of Import Design Screen

Export your list of uploaded changes to Excel by selecting the icon in the top right section of the Import Details table, or click the **Export to CSV** button below the table.

Choose to open or save the file by selecting the corresponding button in the download bar that appears at the bottom of your screen, or click **Cancel** to exit.

A .CSV file is downloaded through your browser containing all of the data shown in the Import Details table (i.e. design change ID, date, time, changed by, description, notes, and status), and also includes any other fields that are included in the XML of the change record, but are not displayed in the on-screen table.

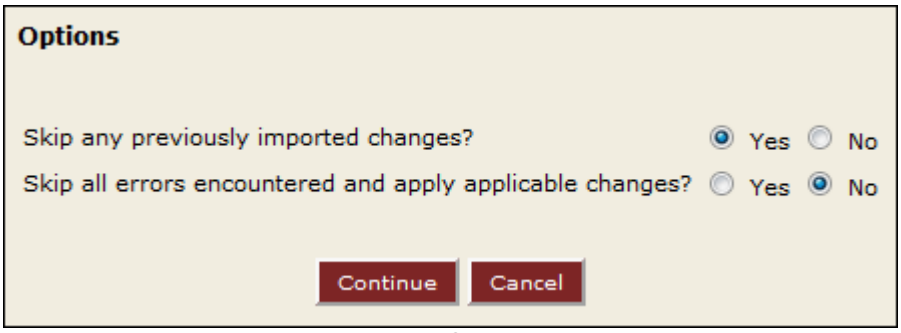


Export to Excel Icon

Once the data is generated into the Import Details table, click the checkboxes of the changes to be applied, then click **Apply**. All applied changes appear in the Imported Change History table at the bottom of the page.

Important Note: *You cannot stop the import unless an error occurs. If you log out of TeamConnect, the import will continue to change the environment, but will go no further if an error is encountered.*

Upon clicking **Apply**, choose to skip previously imported changes or errors encountered by clicking the "Yes" radio button in the **Options** pop-up window. Previously imported packages are already identified in the **Status** column after file upload. If you choose not to skip errors, you still have the option to skip all errors after the first error is generated during importation. The reason for having multiple options to skip errors is helpful if the imported package has a long list of changes and the first error may not be identified for a while.

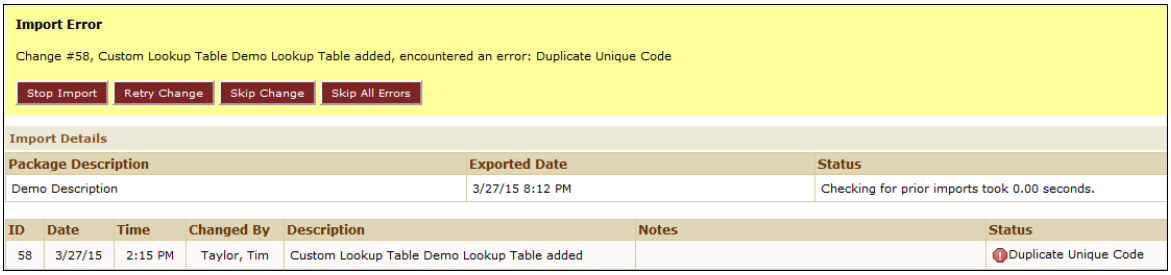


Import Options

Previously imported packages are skipped without stopping the import process as an error might. In order to eliminate duplicate packages, any package that has already been imported displays the information in the second **Status** column (see example below).

Error notices help you identify a solution for the issue, and TeamConnect provides warnings for the following cases:

- A change has already been imported (duplicate ID and data).
- There is an ID gap between the last imported change and the first change of the uploaded package (this may have occurred when selecting design changes to add to the Export List).
- Any of the uploaded design changes were done on a TeamConnect version that is later than the current instance—***Note:*** *TeamConnect allows the import of design changes that were done on an older TeamConnect version. The error serves as notice in case this was done unintentionally.*



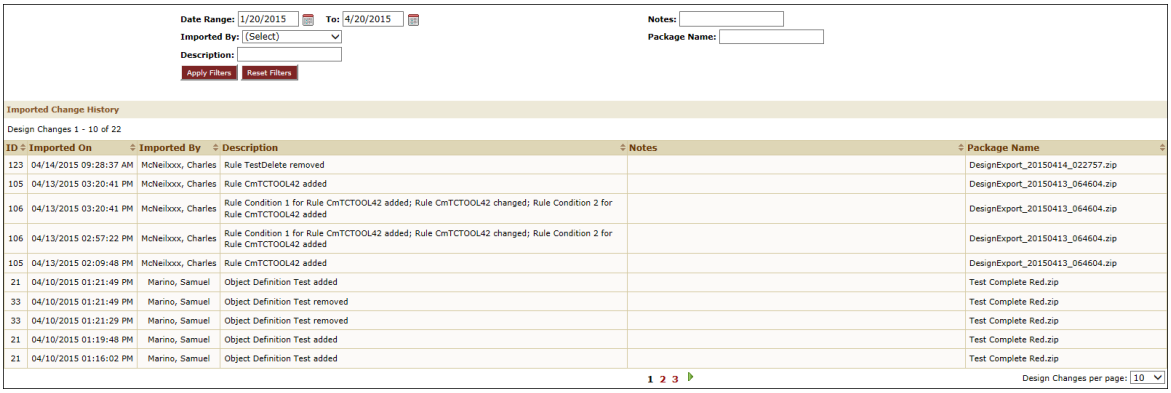
Import Error Options

At this point, you have several options:

- **Stop Import:** Cancel design import and start over.
- **Retry Change:** Reattempt the import if you have made the necessary changes.
- **Skip Change:** If the Import Details table has more than one item within the package, you can select this option to skip the first error to test the remaining items one-by-one. Selecting **Skip Change** does not import failed changes.
- **Skip All Errors:** If the Import Details table has more than one item within the package, you can select this option to test all items simultaneously. Selecting **Skip All Errors** does not import a failed change.

1.1.16.4.2.1 Imported Change History

At the bottom of the **Import Design Changes** page, you can view all imported packages in the Imported Change History table.



Imported Change History

Use the search filter above the Imported Change History table to pinpoint a specific package import, or to view changes made during a specific time range. Click the icons to select "from" and "to" dates, or type a date directly into the date fields.

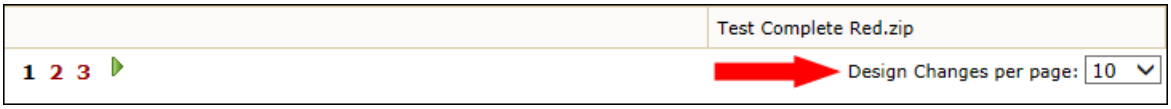
1. Find changes made by an individual by selecting a TeamConnect user from the Changed By drop-down list, or search by description of the imported package by typing directly into the Description text box.
2. If the imported package contained text in the **Notes** column, you can search for that package by matching the text in the Notes text box.

3. Click **Apply Filters** to search using the updated criteria, or **Reset Filters** to start over. Results display above the search filter.

4. Further refine your search by toggling the direction of the lists using the directional arrows ( ) in each of the column headers. The first click causes the column to list items in ascending order; click again for descending order.

At the bottom of the Imported Change History table, dictate how many design changes you would like to see per page by making a selection from the drop-down list (shown below).

Navigate through multiple pages of design changes using the number hyperlinks or the icon to the left of the drop-down.



1.1.16.4.3 Troubleshooting and Best Practices

- When creating a Custom Action or Scheduled Action Rule (Object Definition>Rules) within setup for an Address Book as the recipient, the address book must first be created in the import instance before the actions assigned to the address book can be successfully imported into the new instance. To create the address book, click the **Contacts** tab at the top of the TeamConnect application, then select the **New** drop-down and click **New Address Book**. Enter the same name as the address book for which the custom action or scheduled action rule is being created, then click **Save**.

- Workflow: Make configuration changes only on development server(s). If a problem is discovered in staging or production, correct this problem in development, re-export the changes, then import the changes again. This process takes more time, but it ensures that the same content is deployed and tested across all environments.

- Export: Build and test small export packages before assembling large ones. Once you can verify that they are importing accurately, you can consider bundling them into larger packages for export.

- Import: Skip errors when importing to a non-production environment. It is more efficient to troubleshoot errors found during import after completion, rather than stopping at every error. Refer to the results and warning icons to locate the steps that failed, and test again once they are fixed. However, when importing changes in production, it is best practice to stop immediately when an error occurs, as errors should generally not be expected for this type of environment.

- Users: Log in as a "Special User" to avoid the possibility of two users importing at the same time. The **Design Change Tool** does not prevent two users from simultaneously performing imports on the same server, which could result in unpredictable behavior and conflicting changes. It is best practice to define a design change group and assign a single user with permissions to the design import section of the application. If you have a package that is in the process of importing changes and another user attempts to import changes, both imports will fail.

- If a design change has dependencies (or rules on which a custom object depends) on artifacts on the **Export** instance, but not on the **Import** instance, the dependencies will need to be in the same package. If any package is imported without the dependencies either existing in the **Import** instance's design, or in the package itself, the import process will result in an error.

- If your search results preferences is set to display less records per page than the number of design changes you have made, the changes may not appear in their entirety. Avoid this by going to **Preferences** and entering a larger number (i.e. 999) in the "Number of records per result page" text box, located under Search Results.

A brief description of the XML package tags:

Tag	Description
Operation type	The design change will be either an update, insertion, or deletion.
number	This is the display order of the change.
entityName	The type of change (e.g. rule condition, object data field, user system setting, etc.).
exportDate	The date and time the export was committed.
description	This is the description the user entered for the package.
Change description	A short description of the design change included in the package. This is the same description found on the **Export Design Changes** page in the **Description** column.
type	The category that the design change falls into. This is the same description found on the **Export Design Changes** page in the **Type** column.
time	The time the design change was made.
tcVersion	The TeamConnect application version (update) on which the change was made.
objUniqueCode	A unique code required for identification that was entered by the user at the time of design change creation. This code is related to an object and its object rights, and ensures that proper rights are assigned to the correct object, despite order of import or object creation.
notes	Any notes appended to the change by the user appear here.
id	The ID number of the design change as it appears on the **Export Design Change** page.
date	The date on which the design change was made.

Tag	Description
changedBy	The name of the user who made the design change.

1.1.16.4.4 Artifacts Captured for Custom Designs

At the bottom of this page is a table outlining the custom design changes that are captured in the TeamConnect Configuration Transfer Utility environment. Custom design changes are made in TeamConnect Setup.

Note: *CTU does not capture Reporting artifacts.*

Click on the highlighted link for more information on the artifact and how to make captures for exporting and importing.

Artifact	Category
Assignee Roles	Object Definition XXXX
Audit Appenders	Audit Appender
Categories	Object Definition XXXX
Custom Blocks	Object Definition XXXX
Custom Fields	Object Definition XXXX: Custom Field
Custom Look-up Tables	Custom Look-up Table
Custom Look-up Table Items	Custom Look-up Table
Custom Tools	Custom Tool
Default object views	Settings
Global Navigation	Global Navigation
Group accounts	Group Account
Group account tool access	Group Account

Artifact	Category
Group account object copy items	Group Account
Home pages	Home Page
Home page panes	Home Page
Home page pane content items	Home Page
Line item adjustment reasons	Object Definition $LNI
Line item display items	Object Definition $LNI
Line item expense unit price items	Object Definition $LNI
Line item task codes	Object Definition $LNI
Notification templates	Notifications
Object definitions	Object Definition XXXX
Object definition display attributes	Object Definition XXXX
Object definition files (automated actions)	Object Definition XXXX: Automated Action
Object definition files (automated qualifiers)	Object Definition XXXX: Automated Qualifier
Object definitions files (screens)	Object Definition XXXX: Screen
Object definition unique attributes	Object Definition XXXX
Object definition phases	Object Definition XXXX
Object definition phase transitions	Object Definition XXXX
Object definition batch view items	Object Definition XXXX
Object definition field configuration	Object Definition XXXX
Object definition custom messages	Object Definition XXXX

Artifact	Category
Object views	Object Definition XXXX: Object View
Object view tabs	Object Definition XXXX: Object View
Object view tab blocks	Object Definition XXXX: Object View
Portal panes	Portal Pane
Portal pane content items	Portal Pane
Routes	Route
Route stops	Route
Route stop members	Route
Route notifications	Route
Rules	Object Definition XXXX: Rule
Rule action parameters	Object Definition XXXX: Rule
Rule qualifier parameters	Object Definition XXXX: Rule
Rule conditions	Object Definition XXXX: Rule
Rule history description items	Object Definition XXXX: Rule
Rule history fields	Object Definition XXXX: Rule
Search Views	Object Definition XXXX: Search View
Search view criteria items	Object Definition XXXX: Search View
Search view result items	Object Definition XXXX: Search View
System appenders	System Appender
System custom messages	Settings

Artifact	Category
System folders (every folder under System except Design Versions and every folder under HTTPRoot)	System File
Templates	Object Definition XXXX : Template
Template fields	Object Definition XXXX : Template
Template sub templates	Object Definition XXXX : Template
Template sub template fields	Object Definition XXXX : Template
Template related templates	Object Definition XXXX : Template
Template categories	Object Definition XXXX : Template
Wizards	Object Definition XXXX : Wizard
Wizard rule condition	Object Definition XXXX : Wizard
Wizard pages	Object Definition XXXX : Wizard
Wizard page components	Object Definition XXXX : Wizard
Wizard page actions	Object Definition XXXX : Wizard
Wizard page rules	Object Definition XXXX : Wizard

1.1.16.5 Troubleshooting

This section is organized into the following areas:

- Warnings
- General Troubleshooting Steps
- Error Messages and Performance Issues

1.1.16.5.1 Warnings

This section provides general information about some elements and objects that can be imported using the Design Import tool.

- Design Import Restrictions

- Source Design With No Design Component

1.1.16.5.1.1 Design Import Restrictions

- Currently, the design upgrade file is created only for updating a TeamConnect instance design from an older design to newer design. If you need to revert a QA or production TeamConnect instance to an older design version, there are two possible methods:
  - If the older design version is captured in an earlier snapshot, use the method described in Reverting to a Previous Snapshot.
  - If there is no earlier snapshot, you must make the design changes manually on the development TeamConnect instance, take a snapshot of the current design, and create a corresponding design upgrade file.
- This release of the Design Import tool does not support importing user accounts.
- Deleting System Settings is not supported during importing.
- Please be aware that when you import specific object types, some child elements are deleted from the destination TeamConnect (QA or production) and replaced by the child elements from the source TeamConnect design (from the development instance).

Please see the following table for specific details:

When Importing and Updating the Following Objects:	The Following Logic Will Apply:
Route stops	The source design's memberList child elements will replace those of the destination design.
Home pages	The source design's contentList child elements will replace those of the destination design.
Portal panes	The source design's contentList child elements will replace those of the destination design.
Rules	The source design's conditionList, historyDescriptionList, and historyFieldList child elements will replace those of the destination design.
Wizards	The source design's pageList and child elements will replace those of the destination design.

1.1.16.5.1.2 Source Design With No Design Component

Generally, if the source design (from the development TeamConnect instance) does not include a design component (defined at the XML element level) and the destination design includes the corresponding design component, the import deletes that component from the destination TeamConnect database (for example, QA or production TeamConnect).

For example, if a Dispute object definition is being imported and the source XML file contains definitions for four phases, and the destination XML file contains definitions for five phases (including the four phases from the source database plus an additional phase), the import results in the deletion of the unique phase from the TeamConnect destination database.

This deletion would appear in the log file, but that log entry is not tagged with any specific error message.

1.1.16.5.2 General Troubleshooting Steps

This section provides a general flow for locating the source of DMT issues within a design "diff." In addition, a few examples of errors you could find in the DMT log file and troubleshooting tips are provided.

1. Click the **View Log** link to open the dmt.log file.
2. If a DMT operation is successful, the last line of the log file will contain a success message. Otherwise, the end of the log file will contain a message that the operation failed, followed by detailed messages about each error.
3. Search for "FATAL" in the log to find issues that must be resolved for the import (or snapshot creation or upgrade file generation) to complete.
4. Afterward, search for "ERROR" in the log for additional issues.

1.1.16.5.2.1 Logging

DMT creates a new log file any time it begins an import, export, or diff operation. The name of the log file is always **dmt.log**. Any previous log file is renamed to

```
dmt.import.log.YYYY-MM-DD-HH-MM
```

where the file extension represents the date and time the previous file was last modified. (Note that the string `import` in the example above may instead say `export` or `diff`, depending upon the operation that was performed.) DMT retains up to 10 log files. The oldest file is deleted when creating a new one would exceed the limit of 10.

1.1.16.5.3 Sample Error Messages and Tips

The following are examples of error messages you might find in the log file. Specific details may differ for your designs/diffs but you can apply these to troubleshooting concepts for your specific scenarios.

- **Error message:**

```
FATAL - ImExCustomLookupTables.import: caused exception importing custom lookup
table=LOOK1UP.xml com.mitratech.teamconnect.foundation.TCException: Cannot
delete YDetailLookupItem because it still has 4 EProjDetailObjeValues
referencing it
```

a. To locate the corresponding *.XML source file, from your diff path, navigate to:

diff\Lookup Tables\Custom*LOOK1UP*.xml

where *diff* is the path to the diff

If the issue were in a system lookup table, you could find the *.XML file in *diff*\Lookup Tables\System*LOOK1UP*.xml

b. If you open the source *.XML file, the first line should end like:

```
"opcode="delete"
```

This is the command that tries to delete lookup table items.

c. It is recommended to rename the *.XML file (like *.XML.old) so the DMT will skip the file during import but you can later track changes that might need to be made manually.

- **Error message:**

```
FATAL - ImExCategory.import: exception unmarshalling.Please enter a unique
name.
com.mitratech.teamconnect.foundation.TCException: Please enter a unique name.
```

a. To locate the corresponding *.XML source file, look in the dmt.log file at the events just above (before) the FATAL message. A sample excerpt follows:

```
DEBUG - ImExCategory.importCategory: key=OTHE
DEBUG - ImExCategory.importCategory: itemKey=PR01_OTHE, opCode=insert
DEBUG - ImExCategory.importCategory: inserting category for
itemKey=PR01_OTHE, parent=PR01
DEBUG - ImExCategory.importCategory: inserting category for itemName=OTHER,
key=OTHE, order=0
```

b. The first four characters of the itemKey, PRO1_OTHE, or **PRO1**, indicates the object for which a duplicate category was found.

c. From your diff path, navigate to:

diff\Object Definitions*PRO1*\categories.xml

where *diff* is the path to the diff

d. Open the categories.xml file and search for "PR01_OTHE"

For example, the context could look like:

```
<Category displayOrder="" name="Other" treePosition="PR01_OTHE"
isActive="true" opcode="insert" />
```

e. Generally you must verify that you do not create a category that already exists in the target instance. It is recommended to comment out the corresponding line from the categories.xml file so DMT will skip insertion of the duplicate category.

- **Error message:**

```
FATAL - ImExCustomFields.import: caused exception for filename=PR01_0112.xml
com.mitratech.teamconnect.foundation.TCException: Cannot delete Detail Field
because it still has 1 EProjDetailTextValue referencing it
```

a. To locate the corresponding *.XML source file, from your diff path, navigate to:

diff\Object Definitions\PRO1\Custom Fields*PR01_0112*.xml

where *diff* is the path to the diff

b. If you open the source *.XML file, the first line should end like:

```
opcode="delete"
```

This is the command that tries to delete the detail field.

c. It is recommended to rename the *.XML file like *.XML.old so the DMT will skip the file during import but you can track changes that might need to be made manually.

1.1.16.5.4 Error Messages and Performance Issues

This section contains some limitations in the Design Snapshot tool or Design Import tool. For most of these issues, workarounds are provided. If a workaround is not provided and you need more information, please contact Mitratech support.

- BUILD FAILED message
- DMT script errors

1.1.16.5.4.1 BUILD FAILED message

Issue: An attempt to create a design snapshot, generate a design upgrade file, or import a design upgrade file results in an error message.

Workaround: If you are creating design snapshots from, or importing a design upgrade file to, a TeamConnect instance using an Oracle database, verify that you have the latest ojdbc driver file from Oracle in the DMT **\lib** directory. Also verify that your class path (environment variable) is not pointing to another directory that might contain a different version of the driver (for example, **C:\j2sdk1.5.0_6\jre\lib\ext**).

1.1.16.5.4.2 DMT script errors

Issue: If you are troubleshooting a Design Snapshot tool or Design Import tool operation, and the log contains error messages like the following:

```
java.sql.SQLException: Data size bigger than max size for this type: 23862
at oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:134)
at oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:179)
```

Workaround: Find the latest database driver. If you are using an Oracle thin JDBC driver, you may receive a similar error message when trying to import a design that is 4KB or larger. If you are using an Oracle OCI driver and try to import a design that is 4KB or larger, the entire design will not be imported successfully resulting in functionality limitations.

1.1.17 About Document Generator

Document Generator is a tool that automatically generates customized letters and other documents from XML templates. You may design these documents according to your organization's needs and requirements.

Your TeamConnect users may have the need to write various types of documents on a frequent basis. For example, in the insurance industry, correspondence such as a claim notification needs to be sent whenever an insured client is involved in an automobile accident. Most likely, the same letter always needs to be sent, with certain information in the letter being entered specific to the client.

For another example, information such as addresses, involved parties, and possible violations of the law are different for each client. Instead of having to manually type the same document each time a

client is involved in an accident, a user can create customized letters from a template located in TeamConnect's Documents area.

To become a proficient document template designer, you need to invest some time in learning and understanding how the tool works and what it needs in order to generate a letter.

1.1.17.1 Document Generator Basics

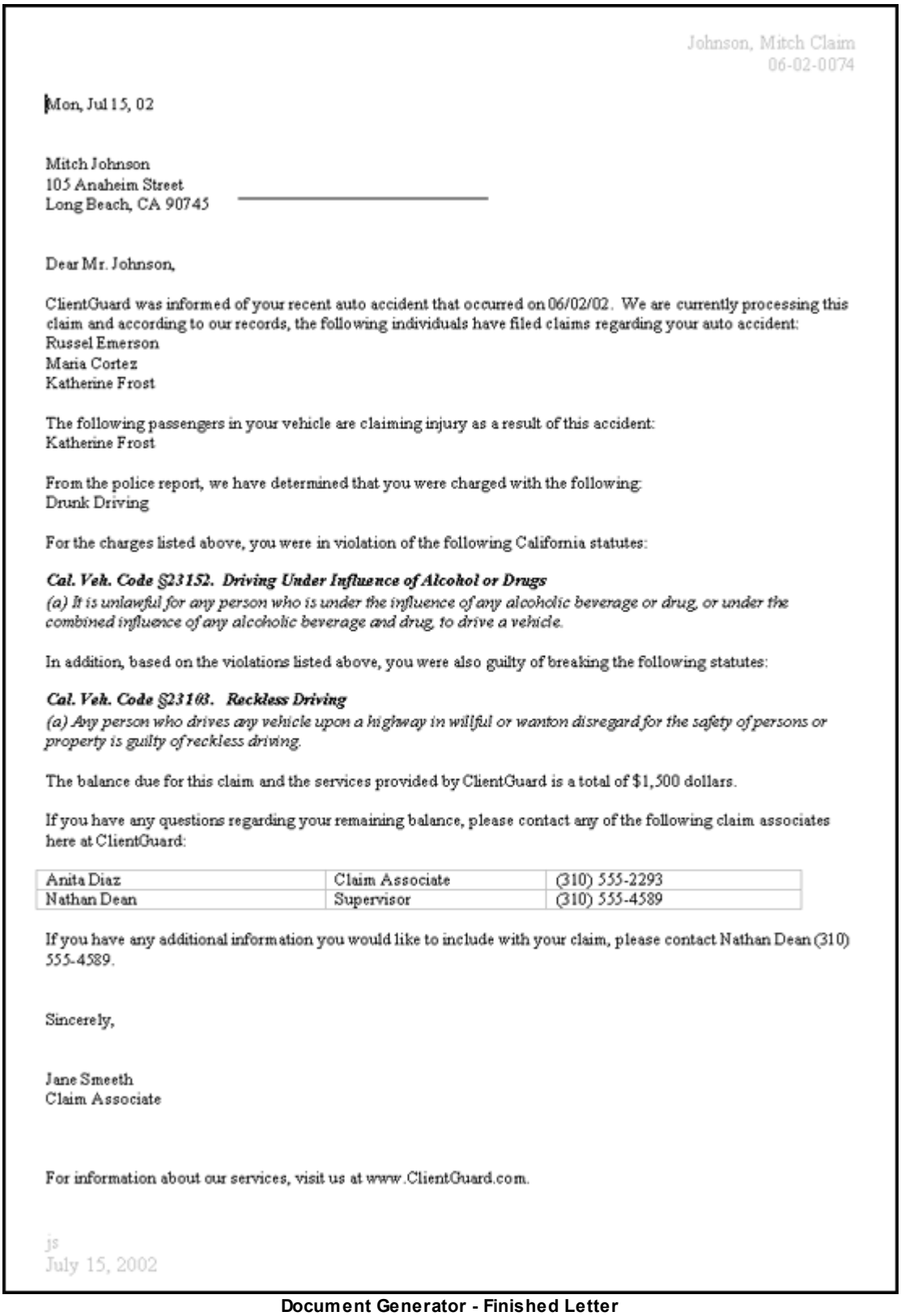
Document templates give your users a quick and easy way to generate documents using data from TeamConnect. In order to create a document template, you must learn how to use the various tags available to automatically retrieve the correct data.

Important: *In your design process, keep in mind that, while using Document Generator, the user must finish working with the document created during the current session. If the user navigates away from Document Generator before finishing, all work done on the document is lost.*

You may even define screens in which the user selects relevant data and manually enters values when generating a document. This definition becomes a part of a document template.

Generated Letter Example

The following image is an example of a finished letter that was created by a user with the Document Generator. Everything from the date, to the salutation, to the Contact phone numbers, are dependent upon a document template.



Johnson, Mitch Claim
06-02-0074

Mon, Jul 15, 02

Mitch Johnson
105 Anaheim Street
Long Beach, CA 90745

Dear Mr. Johnson,

ClientGuard was informed of your recent auto accident that occurred on 06/02/02. We are currently processing this claim and according to our records, the following individuals have filed claims regarding your auto accident:
Russel Emerson
Maria Cortez
Katherine Frost

The following passengers in your vehicle are claiming injury as a result of this accident:
Katherine Frost

From the police report, we have determined that you were charged with the following:
Drunk Driving

For the charges listed above, you were in violation of the following California statutes:

Cal. Veh. Code §23152. Driving Under Influence of Alcohol or Drugs
(a) It is unlawful for any person who is under the influence of any alcoholic beverage or drug, or under the combined influence of any alcoholic beverage and drug, to drive a vehicle.

In addition, based on the violations listed above, you were also guilty of breaking the following statutes:

Cal. Veh. Code §23103. Reckless Driving
(a) Any person who drives any vehicle upon a highway in willful or wanton disregard for the safety of persons or property is guilty of reckless driving.

The balance due for this claim and the services provided by ClientGuard is a total of $1,500 dollars.

If you have any questions regarding your remaining balance, please contact any of the following claim associates here at ClientGuard:

Anita Diaz	Claim Associate	(310) 555-2293
Nathan Dean	Supervisor	(310) 555-4589

If you have any additional information you would like to include with your claim, please contact Nathan Dean (310) 555-4589.

Sincerely,

Jane Smeeth
Claim Associate

For information about our services, visit us at www.ClientGuard.com.

js
July 15, 2002

Document Generator - Finished Letter

Object Model Structure for Identifying Data

It is important to know the areas in TeamConnect from where Document Generator can retrieve information. Knowing where to retrieve information, and the object for which you are creating the template, helps you select the appropriate fields, sub-objects, related objects, XML files, and so on, in order to design a functional document template.

TeamConnect's Object Model is your guide for identifying data that must be retrieved by your template. TeamConnect data is organized in a structure called the Object Model. Each object in the Object Model has its own attributes, and the objects have specific relationships with one another through certain attributes.

When you construct the tags that identify the data that will be inserted into generated documents, you use the relationships and object attribute names that are found in the Object Model to "navigate" to the data you need.

See Object Model: Read This First and the reference tables in later appendixes, for explanations of each attribute in the Object Model. You may use these references to help you construct the tags in your template. The reference tables familiarize you with the relationships between objects.

You may also find it helpful to become familiar with the Object Model by exploring it using the Object Navigator tool. For more details about using Object Navigator, see Using Object Navigator.

Object Dependency

Each document template used by Document Generator is based on a single object. Upon finishing a document template, you must upload the template into the Document Templates folder of a specific Object Definition within the Documents area of TeamConnect.

The object to which the template belongs is the point of origin for all tags that you use in the template to retrieve data. You must be able to identify the data that you want to be populated in the document through relationships between objects in the Object Model.

Information Retrieval

Depending on what you specify in the document template, Document Generator is capable of retrieving information from the following sources:

- TeamConnect Object Records — Document Generator can get information from custom fields, system fields, sub-objects, related objects, user accounts, and the system date.
- Document Generator User — Document Generator can get information that the user manually enters into input pages or selects from filter pages. These screens appear in the user interface of Document Generator and are created by you within the document template.
- Other XML files — Document Generator can get information found in XML files stored in the Documents area of TeamConnect.

Document Template Structure

The document template is composed of three major sections. Each of these three sections are created with Document Generator tags that tell TeamConnect exactly what information to include in the generated document. The three sections of the document template and their respective tags are:

- Content — Provides the necessary data that appears in the generated document, such as text that you writ, and specific information extracted from TeamConnect system and custom fields.

  The content section of the document template is made up of:

  - Text
  - Document Generator tags

- Header — Provides data that asks the user, through filter and input pages, to make a choice (for example, choose a record, item, or enter information) that will be reflected in the generated document.

  The header section of the document template is composed of the following Document Generator tags:

  - `tc:header`
  - `tc:filter`
  - `tc:input`
  - `tc:inputForm`

- Document — The content and header of the document template are enclosed within the following tag:

  - `tc:document`

    This tag has attributes that specify what application to launch in order to open the generated document, as well as the name of the template as it will appear to the user in Document Generator.

Overview of Template Creation and Usage

When a user creates a document from the Document Generator, the procedure is very simple and straightforward. After invoking the Document Generator, the user is prompted by the user interface of the Document Generator to manually enter information into input pages, such as initials, paragraphs of client-specific information, and so on, and/or to select already existing TeamConnect information from filter pages, such as a Contact's name and address. The information that the user enters/ selects appears in the final generated document.

Document Generator templates are created and put into use in the following way:

1. You (the Solution Developer) write a preliminary version of a template, or a preliminary version of the template is provided to you by an individual who is familiar with the business requirements for the template. This document includes placeholders identifying where data must be dynamically populated when a user generates a document based on the template.

2. You go through the template, replacing each placeholder with actual tags that can be read by the Document Generator, and then upload it to TeamConnect.
3. You test the template to insure that the tags used in the template are retrieving the correct data.
4. When the end user wants to generate a document based on the template, he or she will probably be working with a particular record. The user goes to that record's **Documents** block (or tab), opens the Document Generator tool, selects the template, enters any information required by the template, and clicks a button to generate the document.
5. The Document Generator reads through the template and retrieves the correct data to replace the tags in the template, and generates a document.
6. The user can then view, save, or print the generated document.

1.1.17.2 Creating Templates: Process Outline

There are multiple steps involved in creating the content, header, and document sections of the document template. These steps are in order within this document.

To create a document template

1. Write the document template text, with placeholders for all tags you will need. These placeholders will be replaced later with Document Generator tags. This step is described in detail in Writing Document Template Text.
2. Replace placeholders with Document Generator tags that will retrieve data. This step is described in detail in Document Generator Tags Summary.
3. Convert the file to XML and create user interface filter and input pages in the header section.

   If the document requires the user to select any data or enter text into the document, you must create filter and input pages that will be displayed for the user when Document Generator is invoked, by including the appropriate tc:filter and tc:input tags within the header section of the document template. This step is described in detail in Using Document Template Headers.
4. Specify the application that will launch and open the generated document.

   After completing the previous steps, you will need to finalize the template by enclosing all of the text and tags in the appropriate outer tags. This step is described in detail in Completing the Document Template.
5. Test the XML code of the document template.

   Before uploading the document template, you must test it to see if the XML code is well-formatted and fully functional. Internet Explorer, or any other XML viewing application, can be used for this testing.
6. Upload the document template.

   Upload your document template into the proper folder within the Documents area of TeamConnect. This step is described in detail in Uploading Document Templates.
7. Test the document template within TeamConnect.

After uploading the document template to TeamConnect, test the template to verify that the tags in the template retrieve data correctly by generating a document. This step is described in detail in Testing Document Generator Templates.

1.1.18 Using Document Template Headers

The header section of a document template is where you define filter pages and input pages for the end user. A filter page prompts the user who is generating a document to select records, or possibly sub-objects, from which data should be retrieved. An input page prompts the user to enter information that will be included in the generated document.

These pages are defined using the following tags:

- `tc:filter`

  See Defining Filter Pages.

- `tc:input`

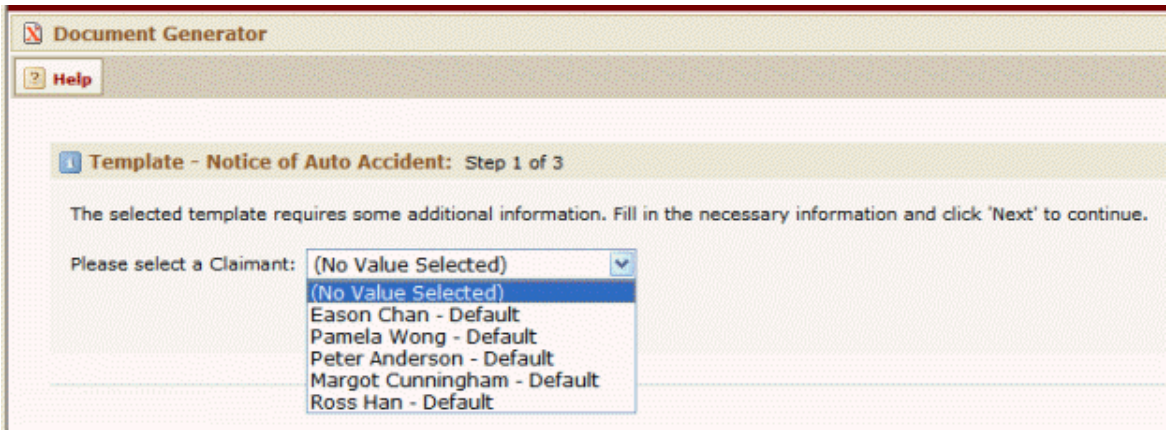
  See Defining Pages for User Input.

The header section of the document template may consist of zero, one or many instances of the tc:filter and tc:input tags, depending on how many different records need to be identified in order to retrieve data, as well as how many different pieces of information the user needs to input manually.

The tc:filter and tc:input tags used in the header section to define filter and input pages must have corresponding tc:filter and tc:input tags in the content section, which refer to the filter and input pages you defined. The tc:filter and tc:input tags in the content section specify the data to be retrieved by TeamConnect.

1.1.18.1 Defining Filter Pages

You must create a filter page to allow Document Generator users to select one or more records containing additional information not available in the current object record. The tc:filter Tag Attributes table shows the result in the end-user interface.



Example: Document Generator - Filter Page

To create a filter page, use the tc:filter tag in the header section of the document template as follows:

- In the header section, the tc:filter tag is used to identify an object record from which the Document Generator should retrieve data.
- Each tc:filter tag that you use in the content section must reference a filter that is defined in the header section.

tc:filter Tag Attributes

The tag attributes noted in the following table are the basic tags you need to create any type of filter page.

In addition to these tag attributes, all tc:filter tags in your header must use at least one other tag attribute to identify an object or sub-object.

tc:filter Tag Attributes

Required ?	Tag attribute	Possible attribute value	Description
Yes	`name`	"FilterName"	Specify a name that uniquely identifies the filter. A name that indicates which object the filter points to is helpful. Important: • This attribute value must exactly match the select="" attribute value of the corresponding tc:filter tag you entered in the content section. You can either define the filter first or write the content section first and define the filter later, but you must use the same name of the filter in both places. • Using spaces in the name of the filter is not recommended.
No	hidden	"yes" or "no"	Indicate whether the filter should be displayed to the user as a filter page, or applied automatically in the background. By default, hidden="no". When hidden="yes", you must provide the attribute list="yes" in the content section of the template where this filter is referenced. For details, see tc:filter.
Yes	`label`	"UserInstructions"	Enter a label that appears as instructions for the user within the filter page.
Yes	`displayString`	"Data;Data;Data"	Specify how the selections appears for the user within the filter page. Multiple data items must be separated by semicolons (;).

			For example, displayString="Contact/FirstName;Contact/Name;DefaultCategory/Name" ***Note:*** *You can only include system fields in the displayString. Custom fields cannot be included.*
Yes	`displayFormat`	"* * - *"	Specify the format of the information displayed by the displayString by typing an asterisk (*) for each data item, and typing any other desired characters, such as hyphens and spaces. Each data item retrieved by the displayString attribute must be represented by an asterisk (*) in the displayFormat attribute.
No	`multi`	"yes" or "no"	Type multi="yes" to allow users to select multiple items simultaneously from within the filter page. By default, only one item may be selected. You do not have to type multi="no" to allow only one selection in the filter page. ***Caution:*** *Do not use multi="yes" when referencing the filter as a main filter. For details, see Main Filters and Sub-Filters.*
Each attribute with an asterisk (*) is required	*searchCondition	searchCondition identifies a related object. For details, see Filters for Related Objects.	
	`qualifier` (used with searchCondition)		
	*test	test identifies a sub-object or a single object record selected in a system field. For details, see Filters for Sub-objects and Single Records.	
	*entity	entity identifies an object that is not related to the main object. For details, see Filters for Non-Related Objects.	
	`criteria` (used with entity)		

No	usedBy	usedBy and objectFrom allow an object identified in one filter to be based on the object identified in another filter. This is a way of navigating to the object record that you want the user to select by using filter pages. For details, see Main Filters and Sub-Filters.
No	objectFrom	

1.1.18.1.1 Filters for Related Objects

When you want a filter page to identify an object that is directly related to either the main object or to an object identified by another filter, use the searchCondition tag attribute of tc:filter. This tag attribute is used to provide the user with a list of related objects.

The tag attributes for identifying related objects are described in the following table. Use these tag attributes in combination with the basic tag attributes in The tc:filter Tag Attributes table.

searchCondition Tag Attribute of tc:filter

Tag attribute	Possible attribute value	Description
searchCondition	"RelatedObject"	Type the name of the related object for which you want to define a filter. Type name of the object as follows: Appointment Account Expense History Involved Task Project (for child custom objects and embedded objects only) ***Note:*** *You must include the qualifier attribute, if you are defining a filter page to retrieve child objects.*
qualifier	"Application_ProjObjectDefinition_UniqueCode=*UCOD*"	This tag attribute is required when you are defining a filter page for a child custom object, child account, or an embedded object. This attribute value identifies the child object using the following values: • Path of the object attribute that uniquely identifies the child or embedded object, which is

		Application_ProjObjectDefinition_UniqueCode. • Unique code of the child or embedded object, which can be copied from the **Object Definition List** in the user interface.

Let's see how the tc:filter tag and its attributes were used in the content and header sections in order to create the filter page shown in the tc:filter Tag Attributes table. In this example, the tc:filter tag is defining a filter page where the user selects a record belonging to the Involved object, which is a related object of the main object Claim.

Notice how the select attribute value in the content section below exactly matches the name attribute value in the header section.

Header

```
<tc:filter name="InvolvedParty" searchCondition="Involved"
hidden="no" label="Please select Claimant" displayString="Contact/
FirstName;Contact/Name;DefaultCategory/Name" displayFormat="* * -
*" />
```

Content

```
<tc:filter select="InvolvedParty">
<tc:data select="Contact/FirstName" />
<tc:data select="Contact/Name" />
</tc:filter>
```

You may need to create a filter page where the user selects a child custom object of the main object. In the following example, the filter allows the user to select a child Claim record belonging to a Policy record (the main object). In the header, the qualifier tag attribute of tc:filter specifies which child object of the Policy object that the filter page must display.

Header

```
<tc:filter name="SelectedClaim" searchCondition="Project"
qualifier="Application_ProjObjectDefinition_UniqueCode=CLAM"
label="Select the appropriate claim"
displayString="Name;NumberString" displayFormat="* - *" />
```

Content

```
<tc:filter select="SelectedClaim">
   <tc:data select="NumberString" />
   <tc:data select="Name" />
</tc:filter>
```

For more examples of how searchCondition can be used, see Main Filters and Sub-Filters. Also, see Document Template Samples.

1.1.18.1.2 Filters for Sub-objects and Single Records

You may want users to select one or more sub-objects from which to retrieve data in a document. For example, you may want the user to select one address from the list of addresses belonging to a contact. In order to provide a filter page where the user selects a sub-object, use the test attribute in the tc:filter tag.

The test attribute can also identify a single record that is selected in a system field. Although you normally do not need a filter page that only gives the user one item to select, this becomes necessary when using tc:filter tags to navigate to the list of related records or sub-objects you want to provide to the user. For example, you may need a filter page where the user selects an Involved (related object) of the parent custom object record Policy (a single record). You will learn more about this in Main Filters and Sub-Filters.

The test tag attribute accepts a path as its value. The path must meet the following requirements:

- It must be constructed of TeamConnect object attributes, according to the relationships between object attributes found in the Object Model.
- It must end with an object attribute that identifies either a single record selected in a system field of a record, or a sub-object list.

The following table describes how to use the test tag attribute for tc:filter. Use this tag attribute in addition to the basic tag attributes noted in the tc:filter Tag Attributes table.

test Tag Attribute of tc:filter

Tag attribute	Possible attribute value	Description
test	"AttributePath" (attributes separated by underscores _)	Use test to specify the path of object attributes separated by underscores (_) that identifies the object attribute that you want the user to select in the filter page. The object attribute identified by the path may be one of the following types: • A sub-object from a sub-object list. ***Tips:*** ○ *Sub-object attributes contain the word "List" in the name of the object attribute (such as AssigneeList, AddressList, or LineItemList).* ○ *Object Model: Read This First (and the additional reference tables it points to) describes all of the sub-objects for each main TeamConnect object.* • An object attribute that identifies a single record (such as the Parent or Contact for a custom object record). Displaying a single record in a filter page is useful when you need to navigate to a list you want the user to view. For more details about this scenario, see Single-Option Filters.

		Tip: *An object attribute that identifies a single record has the data type "object" in this document.*

Sample Values for Test Attribute

If you are identifying a sub-object list with the test attribute, the attribute and value might look like any of the following examples:

- test="AssigneeList" (in a custom object)
- test="AddressList" (in a contact)
- test="Contact_AddressList" (in a contact-centric custom object)
- test="Project_Contact_PhoneList" (in an expense record, to its project's contact-centric field's phone numbers)
- test="Parent_AssigneeList" (in a child custom object, to its parent's assignees)

If you are identifying a single record that is selected in a system field with the test attribute, the attribute and value could look like any of the following examples:

- test="Parent" (in a child custom object)
- test="Parent_Contact" (in a child custom object, to its contact-centric parent)
- test="Contact" (in a custom object, to the contact in its contact-centric field)
- test="Project" (in an expense record)

Header

```
<tc:filter name="UserSelectedAssignee" test="AssigneeList"
displayString="User/Contact/FirstName;User/Contact/Name;Type/Name"
displayFormat="* * - *" label="Please select an assignee." />
```

Content

```
<tc:filter select="UserSelectedAssignee">
    <tc:data select="User/Contact/FirstName" />
    <tc:data select="User/Contact/Name" />
</tc:filter>
```

In the TeamConnect, the filter defined in the header tests the AssigneeList sub-object of the main object and displays a list of assignees in the filter page. In the content section, the first and last name of the assignee is generated in the document.

Header

```
<tc:filter name="SelectedAddress" test="Contact_AddressList"
label="Please select the address of the Insured to which this letter
should be mailed." displayString="Street;City;State;PostalCode"
displayFormat="* *, * *" />
```

Content

```
<tc:filter select="SelectedAddress">
    <tc:data select="Street" />
```

```
        <tc:data select="City" />, <tc:data select="State"/> <tc:data
        select="PostalCode" />
    </tc:filter>
```

In the TeamConnect, the test attribute of tc:filter in the header identifies the list of addresses for the contact selected in the contact-centric field (Insured) of the main object record. In the content section, the address selected by the user appears in the generated document.

Header

```
<tc:filter name="ParentPolicy" test="Parent" label="Please select the
parent Policy below and click Next." displayString="Name"
displayFormat="*" usedBy="1" />
```

In this example of a header only, the filter identifies the Policy selected as the parent in the main object record (the child Claim). This filter will be useful for then allowing the user to select the Policy's Involved parties in a subsequent filter. You will see how this is done in Single-Option Filters.

1.1.18.1.3 Filters for Non-Related Objects

When you want the user to select a record that is unrelated to the main object record, use tc:filter in the header section to define a filter page where the user will make this selection. To create the filter page for an unrelated object, use the tag attributes described in the tc:filter Tag Attributes table as well as those described in the following table.

Filter Tag Attributes - Creating Filter Pages for Non-Related Objects

Tag attribute	Possible attribute value	Description
`entity`	"EntityName"	Type the name of the object from which you want the user to select a record. The user will be able to select from all available records, regardless of whether or not they are related to the main object record. For example, using entity="Account" allows the user to select any available Account. If the entity is a custom object, you must type entity="Project" and use the criteria tag attribute to identify the unique code of the custom object. For example: criteria="objectDefinition='SDTX'".
`criteria`	Use any of the following components in the criteria attribute value: • category='DefaultCategoryName' (the name of the default category of the records you want available in the filter page)	When using the entity tag attribute, you can specify criteria for which object records will be available to the user to select using the criteria tag attribute. This prevents all records in the system from being made available in the drop-down list in the filter page.

	• objectDefinition='ProjectUniqueCode' (if the non-related object is a project, use this component to provide the unique code of the project whose records you want available in the filter page) • name='RecordName' (the name of the record you want available in the filter page) • numberstring='RecordNumber' (the number of the record you want available in the filter page)	You can use any combination of the four components. For each component, you include the value that is desired in order for that record to be available to the user in the filter page. When using the criteria tag components, you must use the following conventions: • Component values are enclosed by single quotes. • Type a comma between components when using more than one. • Do not type any spaces between items or between commas in this tag attribute. • Enclose all components within a set of double quotes, as collectively they are a single attribute value. For example: criteria="category='AutoDriver',objectDefinition='VEHI'"

Header

```
<tc:filter name="SelectAccount" entity="Account"
criteria="category='Office Reserve'" hidden="no" label="Please select
an Account" displayString="Name;Number" displayFormat="* - *" />
```

Content

```
<tc:filter select="SelectAccount">
   <tc:data select="Name" />
   <tc:data select="DefaultCategory/Name" />
</tc:filter>
```

In this example, all Account records from the database that have the default category Office Reserve are listed in the filter page, so that the user generating the document can select an account. In the content section, the name and default category of the account that the user selected are retrieved and included in the generated document.

1.1.18.1.4 Main Filters and Sub-Filters

There are situations where you may want the user to select a record from which data must be retrieved, and this record is not directly related to the main object record. Instead, it is related to a related object of the main object. Or perhaps, you want the user to select a sub-object from a record that is related to the main object. You might even need the user to select a related object or sub-object of a record that is not related to the main object.

For example, if your main object is Policy, you may want the user to select parties Involved (a related object) in the Claim (a child custom object of Policy). This requires two filters: one to specify the Claim and one to specify the Involved.

Another example: you may want the user to select an address (a sub-object) of the contact selected in the Insured field (the contact-centric field) of a Policy (the main object).

These relationships sound complicated but they are not unusual. All of these situations require the use of multiple tc:filter tags, where one filter references another filter to indicate the desired relationship.

These filters are referred to as a main filter and a sub-filter. A sub-filter is a filter that references another filter, called a main filter, to indicate which object owns the sub-object that you are identifying.

Points To Remember

You must know the following points about main filters and sub-filters:

- The main filter must be written before all of its sub-filters in the header section, so that the user first selects the record on which the sub-filters are based.
- The main filter must allow the user to only select one record. You cannot use the multi="yes" tag attribute in the main filter tag.
- The main filter can use searchCondition, test, or entity to identify the object record on which the sub-filter is based.
- The sub-filter can use searchCondition or test to identify a related object or a sub-object of the record identified in the main filter.

The following table describes the tag attribute that is used in the main filter. Use this tag attribute in addition to the basic tag attributes noted in the tc:filter Tag Attributes table.

Tag Attribute for Main Filter

Tag attribute	Possible attribute value	Description
`usedBy`	"Integer"	Type the number of sub-filters that link to the main filter. ***Note:*** *When the usedBy attribute is used, the objectFrom attribute must be used in a sub-filter.*

The following table describes the attribute needed for tc:filter in the header section when defining the sub-filter. Use this tag attribute in addition to the basic tag attributes noted in the tc:filter Tag Attributes table.

Tag Attribute for Sub-Filter

Tag attribute	Possible attribute value	Description
objectFrom	"MainFilterName"	Type the exact name that you used for the main filter.

Related Object of a Related Object (Involved parties of Claim)

In this example, notice how the searchCondition of the main filter indicates which object it is pointing to: the Claim records of the Policy. The sub-filter is referencing the main filter, and the searchCondition attribute of the sub-filter is pointing to the Involved Parties of the Claim.

Header **Main filter**

```
<tc:filter name="SelectedClaim" searchCondition="Project"
qualifier="Application_ProjObjectDefinition_UniqueCode=CLAM"
label="Select the appropriate claim"
displayString="Name;NumberString" displayFormat="* - *"
usedBy="1" />
```

Sub-filter

```
<tc:filter name="ClaimInvolvedParties"
searchCondition="Involved" label="Please select the parties
involved in this Claim" displayString="Contact/
FirstName;Contact/Name;DefaultCategory/Name" displayFormat="*
* - *" multi="yes" objectFrom="SelectedClaim"/>
```

Content

```
<tc:filter select="SelectedClaim">
    <tc:data select="NumberString" />
    <tc:data select="Name" />
</tc:filter>
<tc:filter select="SelectedInvolvedParties">
    <tc:data select="Contact">
    <tc:data select="FirstName" />
    <tc:data select="Name" />
    </tc:data>
    <tc:data select="DefaultCategory/Name" />
</tc:filter>
```

Sub-Object of a Related Object (Addresses of Involved Contact)

In this example, notice how the searchCondition of the main filter indicates which object it is pointing to: the Contact records of the Involveds. The sub-filter is referencing the main filter, and the test attribute of the sub-filter is pointing to the AddressList of the Contacts.

Header **Filter**

```
<tc:filter name="SelectedInvolvedContact"
searchCondition="Involved_Contact"  usedBy="1"
displayString="FirstName;Name" label="Please select an
involved" />
```

Sub-filter

```
<tc:filter name="InvolvedContactAddress"
objectFrom="SelectedInvolvedContact" test="AddressList"
displayString="Type/Name;Street;City" displayFormat="* - *
*" label="Please Select the Involved's Address." />
```

Content

```
<tc:filter select="SelectedInvolvedContact">
    <tc:data select="FirstName" />
    <tc:data select="Name" />
</tc:filter>
```

```
<tc:filter select="InvolvedContactAddress">
   <tc:data select="Type/Name" />
   <tc:data select="Street" />
   <tc:data select="City" />
</tc:filter>
```

Single-Option Filters

In certain circumstances, you will need to provide a filter page that only gives the user one record to select. This is necessary when you want the user to make a selection from a related object or sub-object of an object that is not the main object. For example, suppose you have a contact-centric custom object, Claim, and you want the user to select an address belonging to the contact that is selected in the contact-centric field for the Claim.

In this situation, you have to provide the user with a filter page where the user selects the contact record. This is the main filter, and it must be the only available option. In the next filter page, the sub-filter, the user selects the desired address from the list of addresses for that contact.

The main filter, which only provides one selection to the user, uses the test tag attribute of tc:filter to identify a single record, such as:

- The parent custom object
- The contact in the contact-centric field
- The custom object record with which a task record is associated

A scenario like this is demonstrated in the following example:

Header **Filter**

```
<tc:filter name="RetrieveParent" test="Parent"
label="Please select the parent policy below."
displayString="Name" displayFormat="*" usedBy="1" />
```

Sub-filter

```
<tc:filter name="SelectedInvolveds"
searchCondition="Involved" label="Please select the parties
involved in the parent policy" displayString="Contact/
FirstName;Contact/Name;DefaultCategory/Name"
displayFormat="* * - *" objectFrom="RetrieveParent" />
```

Content

```
<tc:filter select="SelectedInvolveds">
   <tc:data select="Contact">
   First Name: <tc:data select="FirstName" />
   Last Name: <tc:data select="Name" /></tc:data>
   Role: <tc:data select="DefaultCategory/Name" />
</tc:filter>
```

In TeamConnect, the sub-filter indicates that it can display the Involved records of the parent Policy selected by the user in the main filter. In the content section, information about the Involved party, which the user selected in the filter page, appears in the generated document.

1.1.18.2 Defining Pages for User Input

Whenever you need to include information in a document that must be typed by the user who is generating the document, you must use an input tag. In the end-user interface of Document Generator, tc:input tags nested within the header section of the document template define input fields. These fields request text information from the user, which can then be placed directly into the generated document.

For example, your users may need to generate letters in which a paragraph explaining the status of a claim must be typed by the user, while the rest of the letter is automatically generated. In this case, you have to create an input page where the user types the explanation for the generated document.

Note: *When nesting the tc:input or tc:inputForm tags within the tc:header tag, you must make sure that each tc:input tag in the content section corresponds to a tag in the header section.*

The following list shows the different types of input tags you may use in the header section of the document template:

`tc:input` Requires the user to enter text in a single-entry field within one input page in the end-user interface of the Document Generator. For details, see Creating User Input Fields.

`tc:inputForm` Requires the user to enter the appropriate information in multiple-entry fields within one input page in the end-user interface of the Document Generator. For details, see Displaying Multiple Entry Fields in One Input Page.

1.1.18.2.1 Creating User Input Fields

The tc:input tag creates a single entry field within one input page (see the single simple text field example image). Each entry field will appear on a separate page.

The following image provides a list of the attributes and possible attribute values that can be used with the tc:input tag within the header section of the document template.

tc:input attributes in Header Section

Tag attribute	Attribute value	Description
`name`	"FieldName"	Specify a name that uniquely identifies the field you are creating. ***Important:*** *This attribute value must exactly match the select="" attribute value of the corresponding tc:input tag you entered in the content section. The name attribute must always be used when the tc:input tag is nested within the header section.*

label	"FieldLabel"	Enter a label that will display instructions for the user within the input page.
size	"FieldSize"	Specify the maximum size, in characters, of the entry field within the input page. This is the length of the field but does not control the number of characters the user may enter.
maxChars	"MaxNumCharacters"	Specify the maximum number of characters that the user can enter into the input field. ***Note:*** *This attribute is not available when the input field is scrollable (when multi attribute is set to "yes").*
multi	"yes"	Enter "yes" as the attribute value to allow users to enter information into a scrollable text field within the input page (see the single scrollable text field example image). This is useful when a large amount of text needs to be entered by the user. ***Note:*** *By default, users will always enter information into a one-line text field (see the single simple text field example image). If you would like users to have only a text field, simply do not use the multi attribute, there is no need to enter "no" as the attribute value.*
cols	"ColumnWidth"	Specify the width of the columns of the scroll text field. The number you enter represents the number of columns. This attribute is used in conjunction with the multi attribute.
rows	"RowLength"	Specify the length of rows in the scroll text field. The number you enter represents the number of rows. This attribute is used in conjunction with the multi attribute.

Let's see how the tc:input tag and its attributes were used in the content and header sections in order to create the input page with a simple text field shown in the following image. Notice that the select attribute value in the content section exactly matches the name attribute value in the header section.

Header
```
<tc:input name="UserInitials" label="Enter your initials here."
size="15" />
```

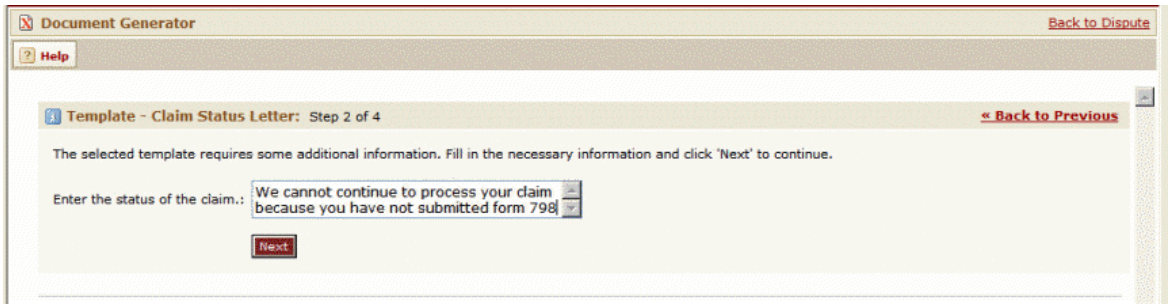
Content
```
<tc:input select="UserInitials" />
```

Example: Document Generator - Input Page - Single Simple Text Field

Next, you may check how the tc:input tag and its attributes were used in the content and header sections in order to create the input page with a scrollable text field shown in the following image.

Header
```
<tc:input name="ClaimStatus" label="Enter the status of the claim."
size="50" multi="yes" cols="45" rows="2"/>
```

Content
```
<tc:input select="ClaimStatus" />
```



Example: Document Generator - Input Page - Single Scrollable Text Field

1.1.18.2.2 Displaying Multiple Entry Fields in One Input Page

The inputForm tag lets you display multiple entry fields within one input page (see the multi-entry field example image for example). This task is accomplished by nesting tc:input tags within a tc:inputForm tag. You can nest multiple tc:input tags as needed. Each tc:input tag can use any of the attributes described in the tc:input attributes in Header Section table.

tc:inputForm attributes in Header Section

Tag attribute	Attribute value	Description
name	"FormName"	Specify a name that uniquely identifies the input page you are creating. • Within the tc:inputForm tag, you must nest each tc:input tag. • In the content section, the value for the select attribute of the corresponding tc:input tag must be the name attributes of both the tc:inputForm and tc:input in the header section, separated by a period (.).

label	"FormLabel"	Enter a label that will display instructions for the user within the input page.

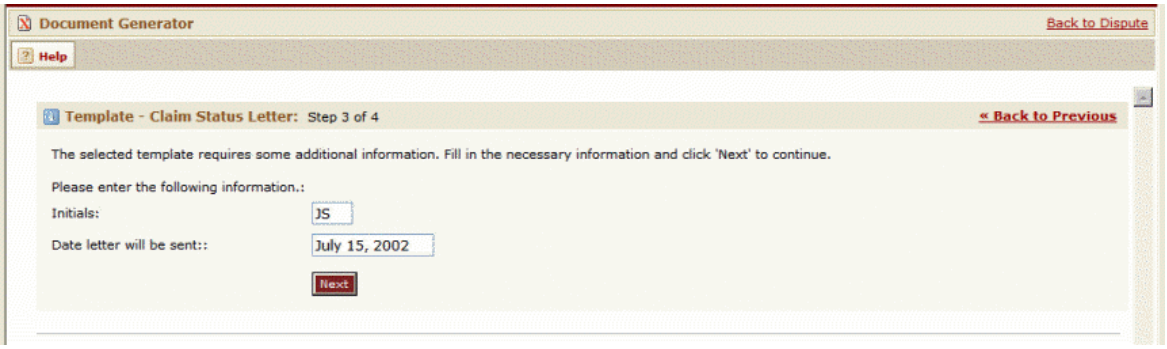
Let's see how the tc:inputForm tag and its attributes were used in the content and header sections in order to create the input page shown in the single simple text field example image. Each tc:input tag is nested within the tc:inputForm tag. Notice that the select attribute value in the content section contains the name attribute value of both the tc:inputForm and tc:input tags in the header section, separated by a period (.).

Header

```
<tc:inputForm name="UserInput" label="Enter the following information">
    <tc:input name="UserInitials" label="Initials" size="3"
    maxChars="3" />
    <tc:input name="LetterDate" label="Date letter will be sent"
    size="15" />
</tc:inputForm>
```

Content

```
<tc:input select="UserInput.UserInitials" />
<tc:input select="UserInput.LetterDate" />
```



Example: Document Generator - Input Page - Multi-Entry Field

1.1.19 Using Document Generator Templates

The content section of a document template consists of the following major components:

Text

The general information that is the same across all documents created from the same template. The text you include in the content of the document template is static information that never changes and appears in the generated document exactly as you typed and formatted it in the original template.

Document Generator Tags

Retrieve data from TeamConnect records. The document template tags you include in the document template are dynamic information that always change according to the data the tags retrieved from TeamConnect records. You learn more about Document Generator tags in Document Generator Tags. The tags are replaced with specific data in the generated document.

1.1.19.1 Writing Document Template Text

The first step in creating a document template is to write the text that will appear in the generated document. This text can be written in almost any word processing program that supports RTF (Rich Text Format). Microsoft Word™ and Corel WordPerfect™ are highly recommended. You can also use Microsoft WordPad™.

About Placeholders

While writing the text of the document template, you should use placeholders within the text to mark for yourself where to insert tags later. This method of developing your document template will help you divide the tasks of composing the document and coding it.

Each placeholder should mark where one piece of data should be inserted. For example, if you want the template to retrieve the first name of an Involved Contact for a Claim project, you would enter a placeholder in your template like this:

```
<First Name of Involved Contact for this Claim>
```

Tips on Formatting and Creating Placeholders

- Create placeholders that identify as specifically as possible what data should be retrieved, especially if you will be delivering the document to another individual for converting it to XML.
- All the formatting that you use in creating the text and placeholders of the document content, including line returns, will be carried over to the generated document. If you enter a line return in a series of nested tags, a line return will be added in the final generated document. For example, in the text and placeholders image, notice how the placeholder tags are on different lines. This is how the retrieved data will appear in the final generated document.
- You can also place tags within the header and footer sections of a Microsoft Word document, and/or within a table, as long as you make sure that the flow of the text in the word processing application places the tags in the correct order.

The following image is an example of text and placeholders that were created in Microsoft Word. This particular template was used in creating the claim notification letter shown in the finished letter image. Notice how the layout of the template is designed according to the desired format of the final document.

<Claim Name>
<Claim Number>

<Date of Letter>

<Insured's First Name> <Insured's Last Name>
<Insured's Street Address>
<Insured's City>, <Insured's State> <Insured's Zip Code>

<Insured's Salutation>

ClientGuard was informed of your recent auto accident that occurred on <Date of Accident>. We are currently processing this claim and according to our records, the following individuals have filed claims regarding your auto accident:
<List all claimants here>

The following passengers in your vehicle are claiming injury as a result of this accident:
<Involved Injured Passenger Contact's First Name> <Involved Injured Passenger Contact's Last Name>

From the police report, we have determined that you were charged with the following:
<Drunk Driving custom field in Claim>
<Speeding custom field in Claim>
<Reckless Driving custom field in Claim>
<Loud Music custom field in Claim>

For the charges listed above, you were in violation of the following California statutes:
<If Drunk Driving is true, import DrunkStatute.xml here>
<If Speeding is true, import SpeedingStatute.xml here>
<If Loud Music is true, import LoudMusicStatute.xml here>

In addition, based on the violations listed above, you were also guilty of breaking the following statutes:
<If any of the above are true, then import the file RecklessStatute.xml here>

The balance due for this claim and the services provided by ClientGuard is a total of <Subtract Amount Paid custom field from Amount Due custom field> dollars.

If you have any questions regarding your remaining balance, please contact any of the following claim associates here at ClientGuard:

<Assignee first name> <Assignee last name>	<Role>	<Phone Number>

If you have any additional information you would like to include with your claim, please contact <Supervisor Assignee first name> <Supervisor Assignee last name> <Phone Number>

Sincerely,

<User who created letter>
<User Title>

<Any additional comments>

<Initials of user who created letter – input by user>
<Date letter will be sent – input by user>

Document Generator Template - Text and Placeholders

1.1.19.2 Document Generator Tags

Document Generator tags are the tags used to extract data from existing TeamConnect records. You will replace the placeholders you created in the content section (see the text and placeholders image) with the actual Document Generator tags. These tags tell the Document Generator exactly what data to retrieve.

The following points will help you understand and work with Document Generator tags:

- All Document Generator tags begin with the prefix tc:.
- Tags, tag attributes, and tag attribute values are case sensitive.
- Within a tag, always capitalize the first letter of each object attribute, even though it is not capitalized when you are working in other areas of TeamConnect or with other TeamConnect tools. This is true for all Document Generator tags.
- The tags are designed so that you have a great deal of flexibility in how you want to use them. Any tag can be nested within almost any other tag, depending on what you want to accomplish. This also means that many of the tags of the same type can be nested within each other.
- Standards for writing well-formed XML apply when you are writing Document Generator tags in your document template.

1.1.19.2.1 Document Generator Tags Summary

After you have written the desired text for your document template, you must replace the placeholders you created with the actual Document Generator tags that will retrieve the necessary data.

Document Generator tags can be classified into the following main groups:

- **Tags that navigate**

  Navigating tags help you identify where data should come from by "navigating" through TeamConnect's Object Model.

- **Tags that retrieve data**

  Once you have navigated to the appropriate object in the Object Model using navigation tags, these tags retrieve the data.

The following table summarizes the Document Generator tags that must be used depending on the type of data you would like to include in the generated document.

Document Generator Tags

To retrieve data from	Use these tags	Used to navigate	Used to retrieve data	Where to learn more
System Fields	`tc:data` Use this tag to navigate through system fields or to retrieve data from a system field.	**x**	**x**	tc:data

Custom Fields	`tc:detail` Use this tag to navigate through custom fields or to retrieve data from a custom field.	**x**	**x**	tc:detail
Sub-objects (user-selected)	`tc:filter` Use this tag to navigate to sub-objects which are selected by the user, from which to retrieve data. For example, use this tag to identify an address selected by the user to be included in a document. ***Note:*** *You need to define the filter in the header section. You will learn about the header section later.*	**x**		Filters for Sub-objects and Single Records
Sub-objects (non user-selected)	`tc:loop` Use this tag to navigate to sub-objects. For example, the list of assignees for a project is a sub-object. When you use this tag, the selection of data retrieved is predefined in the template, so the data is identified automatically without intervention by the user.	**x**		tc:loop
Related Objects (user-selected)	`tc:filter` Use this tag to navigate to related objects when you want the user to have the ability to select a related object from which to retrieve data. For example, use this tag to let the user select which Involved of a project to include in a document. ***Note:*** *You will need to define the filter in the header section. You will learn about the header section later.*	**x**		For referencing a filter in the content of the document template: tc:filter For defining filters: Defining Filter Pages
Related Objects (non user-selected)	`tc:search` Use this tag to navigate to related objects. For example, if the main object is a custom object, use it to navigate to	**x**		tc:search

	the Involveds or child objects of the custom object. When you use this tag, the data is identified automatically without intervention by the user because the selection of data retrieved is predefined by you in the template.			
Current Date	`tc:date` Use this tag to get the current system date.		**x**	tc:date
Current User	`tc:user` Use this tag to get information about the current user.		**x**	tc:user
XML Files	`tc:file` Use this tag to import content from a different file into the generated document. You can use this tag to include content only under certain circumstances by using it with the tc:conditional tag.		**x**	tc:file
Conditional Content	`tc:conditional` Use this tag to dynamically change the content of the generated document depending on certain conditions.			tc:conditional
	`tc:block` Use this tag to group conditional tags together so that more than one condition will trigger the same content to be generated.			tc:block
Calculating Two Numerical Values	`tc:compute` and `tc:operand` Use this tag to make a calculation using two number fields or a number field and a value that you provide in the template.			tc:compute tc:operand
End-User Input	`tc:input` Use this tag to include text that is entered by a user in an input page.			tc:input

	Note: *You will need to define the input page in the header section of the document template. You will learn about the header section later.*			

1.1.19.2.2 tc:data

The tc:data tag is used to do both of the following actions:

- Retrieve data from TeamConnect system fields.
- Point the Document Generator to different object records from which to retrieve data by nesting tc:data tags, thereby "navigating" to the system fields you need.

What Can Be Retrieved with tc:data

The tc:data tag can retrieve data from the following areas:

- **Main object record** — Use the tc:data tag independently to retrieve single instances of data from TeamConnect system fields of the main object record.
- **Sub-objects** — Nest the tc:data tag within one of the following tags:
  - tc:loop tag, to automatically retrieve data from sub-objects.
  - tc:filter tag, to have the user select the sub-object from which to retrieve the data.

  These tags retrieve data from TeamConnect system fields of sub-objects, such as project assignees and categories. For more information, see:
  - tc:loop or tc:filter
  - Filters for Sub-objects and Single Records
- **Related objects** — Nest the tc:data tag within one of the following tags:
  - tc:search tag, to automatically retrieve data from related objects.
  - tc:filter tag, to have the user select the related object record from which to retrieve the data.

  These tags retrieve data from TeamConnect system fields of related objects (such as involved, appointment, child custom objects, embedded custom objects and child accounts). For more information, see tc:search and tc:filter.
- **Objects not related to the** — Nest the tc:data tag within the tc:filter tag to retrieve data from objects that are not related to the main object. For more information, see Filters

main object for Non-Related Objects.

- **System object related to current object through a system field** — Nest the tc:data tag within another tc:data tag when a system field relates the current object to a system object, in order to retrieve data from the related system object, such as contact. For an example, see Nesting to Navigate.

- **Custom object record selected in a custom field** — Nest the tc:data tag within the tc:detail tag to retrieve data from system fields of a record selected in a custom field of type Custom Object. For an example, see Nesting to Navigate.

Points To Remember

When using tc:data, you must know the following concepts:

- The tc:data tag will retrieve data from the specified system field of the current object. When the tc:data tag is not nested, it points to data from the main record for which the document is being generated.
- The tc:data tag requires one tag attribute, select. It can also accept several other tag attributes, which are used to format the data item that the tag retrieves.
- When using tc:data to retrieve data, the object attribute you are retrieving with tc:data cannot be of the data type Object. An object is actually a collection of attributes that identify data, so it cannot itself be inserted into a generated document. If you attempt to do this, the generated document will contain a large amount of nonsensical code. If you need specific information from that object, you should nest tc:data tags in order to navigate to the specific data you need.

  Tip: *Object Model: Read This First and the additional reference tables it points to indicate the data type of each object attribute in the **Data Type** column of all tables. You can see which attributes have the data type **Object** and avoid retrieving them in a template.*

- If you need to retrieve data from a custom field, use tc:detail. For more information, see tc:detail.

Required tc:data Tag Attribute

Whether using the tc:data tag on its own to retrieve data from the currently selected record, or if nesting the tc:data tag to retrieve data from sub-objects or related objects, the select attribute must always be used with the tc:data tag.

Required tc:data Tag Attribute

Tag attribute	Attribute value	Description
select	"ObjectAttribute"	Specify the system field from which the Document Generator will retrieve data by typing the name of the object attribute from the Object Model, making sure to capitalize the first letter of the name.

		The object attributes that correspond to all system fields are found in the Object Model: Read This First and in the additional reference tables it points to.

Example:
```
<tc:data select="NumberString" />
```

The select attribute takes a system field's name as its value. In the example above, NumberString is the object attribute that corresponds to the system field, **Number**. Therefore, the data retrieved by the Document Generator will be the number of the record selected by the user.

Optional tc:data and tc:detail Tag Attributes

The following attributes are optional attributes that can be used within the tc:data and tc:detail tag. Each optional attribute is applicable to a specific field type (such as number, date, text). These tags control the format of the data that is generated in the document.

Caution: *If you try to assign an attribute to a field that is not of the appropriate type, TeamConnect is not able to generate the document and displays an error.*

Optional Tag Attributes for tc:data and tc:detail

Tag attribute	Applicable field types	Possible attribute values	Default	Description	Example

`numberFormat`	Number	Examples:
		`#,###.00`
		`#,##0.00`
		`####.##`
		`####.###`
		`####`
			If the format of the digit was specified with #, the number will drop all trailing and unnecessary zeros.
			If the format of the digit was specified with a 0, the digit will be displayed as it was typed.
				Specifies the format of number fields, including the placement of the comma and decimal point.
				Using a 0 instead of # tells the Document Generator to display the digit, even if it is a 0. When generating financial information, you should use a format such as #,##0.00.
					`<tc:data select="Unit Price" numberFormat="#,##0.00" />`

`percentage`	Number	`###%` `##0.0#%`	1/100	Specifies that the number should be displayed as a percentage, and	`<tc:data select="CompletedPercent" percentage="###%" />`

				whether decimals should be included. Using a 0 instead of # tells the Document Generator to display the digit, even if it is a 0.	
dateFormat	Date	See the Possible Values for dateFormat Tag Attribute table.	MM/dd/yyyy	Specifies the display format of date and time values.	<tc:data select="DueOn" dateFormat="dd/MM/yyyy" />
showTime	Date	yes no	no	Shows the time next to the date. Only necessary if the value for dateFormat does not display the time. ***Note:*** *This tag should only be used if dateFormat is not specified.*	<tc:data select="DueOn" showTime="yes" />
timeZone	Date	yes no	no	Shows the time zone (in parenthesis) next to the date. The time zone is from the current user's settings. ***Note:*** *This tag attribute is only necessary if the value for dateFormat does not display the time zone.*	<tc:data select="DueOn" showTime="yes" timeZone="yes" />

case	Text	upper lower	Unless an attribute value is specified, the text appears as you typed it.	Specifies that the text should be displayed in all uppercase or all lowercase letters.	<tc:data select="Location" case="upper" />
true/ false	Check-Box	Any text value.	For true, the custom field label. For false, blank.	Specifies the display values in the document for the true and false states of a check-box.	<tc:data select="IsAutoPost " true="Automatic posting is enabled" false="Automatic Posting is disabled" />

dateFormat Tag Attribute Values

The following values are the available values for the **dateFormat** attribute.

Possible Values for dateFormat Tag Attribute

Value	Example output
yyyy.MM.dd G 'at' hh:mm:ss z	2001.12.08 AD at 03:08:17 PDT
MMMMM d, yyyy	December 8, 2001
EEE, MMM d, yy	Tue, Dec 8, 01
hh:mm:ss	03:08:00
h:mm a	3:08 PM
hh 'o clock' a, zzzz	03 o clock PM, Pacific Daylight Time
K mm a, z	3 08 PM, PDT
yyyy.MMMMM.dd GGG hh.mm aaa	2001.December.08 AD 03:08 PM
MM/dd/yyyy ***Note:*** *This is the default value of this tag attribute.*	12/08/2001
dd/MM/yyyy	08/12/2001

`yyyy/dd/MM`	2001/08/12

The date and time formats can be combined to create the desired result. For example, you can combine MM/dd/yyyy with h:mm a (dateFormat="yyyy/dd/MM h:mm a") to display the date as 12/08/2001 3:08 PM.

Navigating Within tc:data Tag

In tc:data tags, you can navigate throughout the Object Model to retrieve data from other objects without opening and closing separate tags for each object. If you only want to retrieve one data item from an object, rather than multiple items, it may be easier to write one tag that points to the data you need rather than nesting to navigate.

A path to a data item, or object attribute, that exists in a different object is retrieved in a single tag by writing a path to it using the object attributes that serve as relationships between objects. These object attributes are noted in Object Model: Read This First and in the additional reference tables it points to, with arrows (-->).

Follow these guidelines when navigating to an attribute within a tc:data tag:

- Separate each object attribute in the path with a forward slash (/).
- Capitalize the first letter of all object attributes in the path.
- Close the tag to point the Document Generator back to the previous object that it was pointing to.

For example:

`<tc:data select="Parent/Contact/FirstName" />`	Retrieves the first name of the Contact selected for the parent record's contact-centric field.
`<tc:data select="Project/Name" />`	Retrieves the name of the Project record to which a Task record is related.
`<tc:data select="MainAssignee/Type/Name" />`	Retrieves the name of the main assignee's role for a record.
`<tc:data select="MainAssignee/User/Contact/FirstName" />`	Retrieves the first name of the main assignee for a Project record.

1.1.19.2.3 tc:detail

The tc:detail tag retrieves data from a TeamConnect custom field. You can use tc:detail to retrieve custom field values from the currently selected record, or you can nest it, thereby "navigating" to retrieve values from objects that you specify.

Keep the following points in mind when working with this tag:

- To retrieve custom fields of related objects, nest the tc:detail tag within the tc:search or tc:filter tag (for details, see tc:search or tc:filter).
- To retrieve custom fields from a system or custom object to which a custom field links, nest the tc:detail tag within the tc:data tag (for details, see Nesting to Navigate).
- The tc:detail tag uses the same attributes as the tc:data tag for formatting the data retrieved by the tag. See the Optional Tag Attributes for tc:data and tc:detail table for a list of additional attributes.

The following table provides a description of the required attribute of tc:detail.

Required tc:detail Tag Attribute

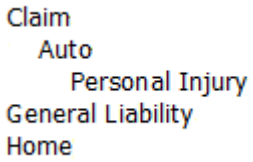
Tag attribute	Attribute value	Description
`select`	`"CategoryTreePosition/FieldName"`	Specify the full category tree position and the field name of the custom field from which TeamConnect will retrieve data. The full category tree position includes the unique code of the object (the root) and the tree position of each category in the tree, separated by underscores (_).

Example: `<tc:detail select="CLAM_AUTO/Personal Injury" />`

In the previous example, **Personal Injury** is a custom field found in a Claim record in the category **Auto**. Since this custom field is a check-box, the label of the check-box is the data extracted and inserted into the generated document if the check-box is checked. Additional categories on the same level are other types of claims, for example, **General Liability** and **Home**.

In this example, CLAM is the unique code of the custom object **Claim**. If you are referencing a custom field in the top node (root) of a custom object's category tree, the tree position of the custom field is the unique code of the object.

The options in the resulting drop-down list in Document Generator appear to the end user as follows:



Custom Field Display Formats

By default, data retrieved from each type of custom field appears in a particular format in the generated document. However, you can modify the way certain data types appear using the tag attributes in the Optional Tag Attributes for tc:data and tc:detail table.

The following table indicates how each type of data will be displayed by default.

Custom Field Display Formats in Generated Document

Custom field type	Display format of values in generated document
Text	Exact data entered in field.
Number	Exact data entered in field.
Date	MM/dd/yyyy
Check-Box	For **true**, the custom field label. For **false**, blank.
Memo Text	Exact data entered in field.
Custom Object	The title of the custom object record. The exact format depends on the **Automatic Numbering Pattern** defined in the custom object definition. This pattern may be any of the following: • Name only • Number only • Name - Number • Number - Name
Involved	The full name of the contact who is the involved, in the following format: `Name, FirstName`
List	The name of the lookup table item.

1.1.19.2.4 tc:loop

The tc:loop tag is used to navigate to sub-objects of the current object. The tc:loop tag does not retrieve any data by itself. It only specifies the sub-object from which TeamConnect will retrieve data. Essentially, it navigates from the current object to the sub-object that you specify.

When you close the tc:loop tag, you return to the object to which the sub-object belongs. However, while inside the tc:loop tag, you can continue to navigate to other objects by nesting additional tags. You can also nest tc:loop within other tags as needed, such as tc:filter or tc:conditional.

To retrieve data from system fields of the sub-object, you must nest the tc:data tag within the tc:loop tag.

When nesting the tc:data tag within the tc:loop tag, the following data can be retrieved:

- Data from all instances of a sub-object.

For example, you may want to include all the names and phone numbers of all the Assignees for an auto claim. For details, see Retrieving Data from All Instances of Sub-object.

- Data from only instances of the sub-object that match a criterion that you define.

  For example, you may want to include only the name and phone number of Assignees with a certain role, such as Supervisor. For details, see Retrieving Data with tc:loop Using Qualifiers.

Automatically Selected vs. User-Selected Sub-objects

The tc:loop tag is useful for selecting instances of a sub-object from which data should be retrieved automatically in a document template. However, there are some cases where it is necessary for the user who is generating a document to select a sub-object from which to retrieve data.

For example, your document template may require that the user select an assignee, address, or phone number. In order to provide the user with a list of instances of a sub-object, you would not use tc:loop. Instead, you would use tc:filter to define a filter page where the user would make the appropriate selection. You will learn more about how to do this in Filters for Sub-objects and Single Records.

1.1.19.2.4.1 Retrieving Data from All Instances of Sub-object

To retrieve data from all instances of a sub-object, regardless of the type, you only need to use the select attribute with the tc:loop tag and then nest the tc:data tag within the tc:loop tag.

For example, you can use this method to list all Assignees for a project, regardless of the assignee role (type). The Document Generator will print the data you specify for all of the assignees of the project.

Usage of tc:loop - Retrieving Data from All Instances of Sub-object

Tag attribute	Attribute value	Description
`select`	`"SubObjectList"`	For the value of the select attribute, type the name of the object attribute that links to the desired sub-object. For example, if you want the Assignees sub-object for a Project, type AssigneeList. See Object Model: Read This First and the additional reference tables it points to, for a complete list of sub-objects and to find the object attribute names that identify them. Typically, attributes that link to sub-objects have the suffix "List" (AssigneeList, PhaseList, and so on). ***Note:*** *The tc:data tag must be nested within the tc:loop tag in order to retrieve any data.*

Example:

```
<tc:loop select="AssigneeList">
<tc:data select="User/Contact/FirstName" />
<tc:data select="User/Contact/Name" />
<tc:data select="Type/Name" />
```

```
<tc:data select="User/Contact/DefaultPhone/PhoneString"/>
</tc:loop>
```

The select attribute of tc:loop takes a sub-object, such as an assignee or a category, as its value. Data values from the system fields of the sub-object specified by tc:loop are retrieved by tc:data tags nested within tc:loop.

In the example given, the data retrieved by TeamConnect is the first and last name, title, and phone number of all the users who are Assignees. The sub-object Assignee is specified by the tc:loop tag. The first and last name, title, and phone number of all the users are specified by the nested tc:data tags. Notice that this information is really stored in the user's Contact record, so the tags are actually navigating to the Contact record.

1.1.19.2.4.2 Retrieving Data with tc:loop Using Qualifiers

To retrieve data from certain instances of the sub-object that meet a certain criterion, rather than all instances of the sub-object, you must use the qualifier attribute with tc:loop. Again, you must also nest the tc:data tag within the tc:loop tag.

For example, you can use this method to retrieve data only from assignees that have the role Supervisor within a project record. If one or more assignees are found to match your qualifier, their data will be retrieved and inserted in the generated document. The Document Generator will automatically retrieve data only from those records that match the criterion, whether there are zero, one, or multiple records that match.

Caution: *The qualifier attribute can only handle one qualifier.*

For most sub-objects, the Type attribute is the most useful characteristic for filtering the data that is retrieved. For assignees, the role is actually the Type attribute of the assignee sub-object in the Object Model.

It is possible to use other object attributes from the sub-object, such as a number or text field, but these values are rarely useful for setting a criterion for which sub-objects to retrieve.

The following table indicates the tags and attributes that you would use to set up a tc:loop that retrieves a qualified list of sub-object data in a document template.

Usage of tc:loop - Retrieving Data with Qualifier

Tag attribute	Attribute value	Description
`select`	`"SubObjectList"`	For the value of the select attribute, type the name of the object attribute that links to the desired sub-object. For example, if you want the Assignees sub-object for a Project, type AssigneeList. See Object Model: Read This First and the additional reference tables it points to, for a complete list of sub-objects and to find the object attribute names that identify them.

		Note: *The tc:data tag must be nested within the tc:loop tag in order to retrieve any data.*
`qualifier`	`"PathToQualifier=Value"`	• PathToQualifier is a path of object attributes, separated by underscores, that qualifies the sub-object list. This path is built according to the logic of the Object Model. The last object attribute in this path must be of one of the following data types: ○ integer ○ string ○ date For example, you can use the type of the sub-object as a qualifier by typing Type_TreePosition for PathToQualifier. The data types can be found in the Object Model: Read This First and related tables. • Value is the data that must exist in the selected attribute in order for the Document Generator to retrieve that instance of the sub-object. If the value does not match the value that you specify, the sub-object will not be retrieved. See Qualifier Tag Attribute Syntax for details about the value of this attribute. ***Note:*** *The Value can be a ; delimited string to provide OR logic. However, only one instance of the sub-object will be returned, rather than all matching instances.*

The following table indicates the syntax for the qualifier attribute value.

Qualifier Tag Attribute Syntax

Object attribute of sub-object used as qualifier	Syntax	Description
Type attribute	`qualifier="Type_TreePosition=UCOD_TREE"`	The Type attribute of a sub-object is the most commonly-used attribute for setting a

		criterion for which sub-objects to retrieve. 1. Replace UCOD with the four-character unique code of the object that TeamConnect will retrieve data from. o If the sub-object is a system lookup table, type the unique code of the system lookup table. o If the sub-object is DetailList (categories), PhaseList, or AssigneeList, type the unique code of the object to which it belongs. This may be found in the **General** tab of the object definition. 2. Replace TREE with the full tree position of the selected item within the sub-object. The tree position should be in all caps, with each level separated with an underscore. For example: `AAAA_BBBB_CCCC` If the desired type is in the Root of the sub-object, then the tree position is simply the four-character unique code of the sub-object.

Any attribute that identifies a number or text value
 	
Note: *Not useful as a criterion.*
	`qualifier="PathToAttribute=ExactValue"`	When using a number or text field to qualify your tc:loop tag, simply type the value that you want to use as the criterion for which sub-objects to retrieve as the Value of the tag attribute. For example, from the sub-object JContAddress:
		`qualifier="State=CA"`
		`qualifier="Version=1"`
		Note: *For number values, you must verify the type of number that is stored in the database and follow that format (float, integer, and so on).*

Any attribute that identifies a date value	`qualifier="PathToAttribute=YYYY-MM-DD"`	Use this syntax for the date value when using a date value to qualify the sub-object. For example, from the sub-object JContRate: `qualifier="EffectiveTo=2003-02-26"`

Note: *Not useful as a criterion.*		

The following examples show how qualifiers can be used in the tc:loop tag.

Example 1:

```
<tc:loop select="AssigneeList"
qualifier="Type_TreePosition=CLAM_SPRV">
    <tc:data select="User/Contact/FirstName" />
    <tc:data select="User/Contact/Name" />
    <tc:data select="User/Contact/DefaultPhone/ PhoneString" />
</tc:loop>
```

In the example above, the qualifier attribute takes the unique code of the object and the tree position of the selected item within the sub-object as its value. CLAM is the unique code of the selected object, Claim, and SPRV is the tree position of the project assignee. The data retrieved by TeamConnect is the first and last name, and phone number of only the type of assignee (specifically, assignee role) that is specified by the qualifier attribute.

Example 2:

```
<tc:loop select="DetailList" qualifier="Category_Parent_Name=Claim">
    <tc:data select="Category/Name" />
</tc:loop>
```

This is an example of using a text value as a qualifier. In this example, which is for a custom object named Claim, the qualifier attribute makes the Document Generator retrieve only those categories added to the record that have Claim as the parent category. This property means that any categories on the level directly beneath Root which are added to the record are retrieved. However, the category Claim itself is not retrieved.

Also, any categories added to the record that are on the second, third, fourth levels, and so on, beneath the Root are not retrieved, because their Parent category is not Claim. The data being retrieved with the tc:data tag is the name of each category that meets the qualifier.

This property is useful when you want the generated document to include only the major categories of a record and not all of the sub-categories.

Example 3:

```
<tc:loop select="PhoneList" qualifier="Type_TreePosition=PHON_BUS1">
    <tc:data select="PhoneString" />
</tc:loop>
```

In the previous example, the qualifier attribute makes the Document Generator retrieve only the phone numbers for a contact that are of the type Business 1 (BUS1). The tc:data tag retrieves the actual phone number.

Notice that the value for the qualifier attribute is the unique code of the lookup table (PHON), followed by an underscore (_) and the tree position of the phone type being retrieved (BUS1).

1.1.19.2.5 tc:date

The tc:date tag retrieves the current system date, at the time the document is generated. By default, the date will be displayed in the following format:

```
MM/dd/yy
```

To display the current system date in another format, use the format tag attribute.

You can also nest the tc:user tag, since it works the same regardless of which other tags you have outside of it. The following table describes the possible tag attributes for tc:date.

Usage of tc:date

Tag attribute	Attribute value	Description
`format`	`"DateAttribute"`	Specify the format in which the date and time will display in the end-user's generated document. See the Possible Values for dateFormat Tag Attribute table for a list of date formats.
`showTime`	"yes" or "no"	Shows the time next to the date. ***Note:*** *This tag should only be used if format is not specified*
`timeZone`	"yes" or "no"	Shows the time zone next to the date. Only necessary if the value for format does not display the time zone. ***Note:*** *The time zone is obtained from the current user's settings.*

Example:

```
<tc:date format="EEE, MMM d, yy"/>
```

In the example given, the date will be inserted in the generated document and formatted by Day, Month Date, Year, as in Friday, Jun 28, 2002.

1.1.19.2.6 tc:user

The tc:user tag retrieves data from the user account of the current user who is generating the document. You can nest tc:data tags to retrieve the necessary data from that user. You can also nest the tc:user tag, since it works the same regardless of which other tags you have outside of it.

This tag points to the YUser object table. See Object Model: Read This First to learn about what data you can retrieve about the current user, either from YUser or from an object that is related to it.

Example:

```
<tc:user>
    <tc:data select="Contact">
    <tc:data select="FirstName" />
```

```
        <tc:data select="Name" />
        <tc:data select="Title" />
        </tc:data>
    </tc:user>
```

In the example given, the data retrieved by TeamConnect is the first name, last name, and job title of the user who generates the document, which is all located in the contact record associated with that user.

1.1.19.2.7 tc:file

The tc:file tag retrieves the text of a separate XML file and places that text in the end-user's generated document. The file you are retrieving must be uploaded to a specific directory in TeamConnect's Documents area. This tag can be nested as needed in a document template.

You can use this tag to retrieve content that is being used by many different templates, instead of including the same content directly in each template. This tag is also useful if the text in the XML file should be included only under certain circumstances. You can use this tag in conjunction with the tc:conditional tag to accomplish this.

Usage of tc:file

Tag attribute	Attribute value	Description
select	"FileFolder/ FileName"	Specify the name of the folder in which the XML file is located and the name of the file. This path is relative to the document template's location in TeamConnect's Documents area. This means that all supporting XML files that are retrieved by a document template must be stored in a subfolder of the **Document Templates** folder where the template is located.

Example: `<tc:file select="Statutes/CalStatutes.xml" />`

The select attribute takes the name of the folder in which the XML file is located and the name of the XML file as its value. In the example given, Statutes is the name of the folder and CalStatutes is the name of the XML file. The content of the CalStatutes file will be included as text in the generated document.

Preparing XML Files Referenced by tc:file

You may wish to retrieve text from a file with a DOC extension (a Microsoft Word™ file) and insert that text into the generated document. However, only an XML file can be extracted with the tc:file tag and inserted into the generated document.

To convert the DOC file into an XML file

1. Open the DOC file with your word processing application (i.e. Microsoft Word™).
2. Format the text exactly as you would like the information to appear in the generated document (enter all returns, indentations, line formatting, font size, and so on).
3. Save the file with an RTF extension and close your word processing application.
4. Find the RTF file you just saved and open it with a text editor such as Microsoft Notepad™.
5. At the very beginning of the document, insert the following tags, with a line return after each, in the following order:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <tc:document version="1.0" mime-type="MimeType" name="TemplateName"
   xmlns:tc="http://www.w3.org/1999/XSL/Transform">
   ```

   For information about the attribute values "MimeType" and "TemplateName" see the Document Tag Attributes table.

   ```
   <tc:content>
   ```

   Caution: *Do not place a line return after tc:content. If the source code does not start immediately after this tag, TeamConnect encounters an error when trying to generate the document.*

6. At the very end of the document, insert the following tags, in the following order:

   ```
   </tc:content>
   </tc:document>
   ```

7. Save the file with an XML extension.
8. Open the file in Internet Explorer or an HTML editor to test whether it is well-formed.
9. Create a new folder within the **Document Templates** folder of the object for which your template is created.
10. Upload the XML file to the folder you just created.

    The contents of the XML file will be included in the generated document as long as the XML file and its folder are in the correct directory in the **Documents** area of TeamConnect.

1.1.19.2.8 tc:conditional

The tc:conditional tag is used to specify a condition that must be met in order for any nested content within the tc:conditional tag, including static text as well as tags, to be included in the generated document. If the condition specified by the tag is met, then anything nested within tc:conditional is generated. If the condition is not met, then the nested content is ignored.

The following points summarize the basics about tc:conditional:

- tc:conditional can be nested within almost any other tags. For example, you can nest it within tc:filter or tc:search. For an example of nesting tc:conditional, see Retrieving Specific Type of Related Object with tc:conditional.
- You can nest tc:conditional tags within each other, if necessary.

- There are two main tag attributes for tc:conditional: compare and test. You must always use either one or the other when you use tc:conditional. However, you cannot use both.
- There are two other tag attributes that you can use in combination with compare or test, which are described in the following table.

The following table describes all of the tag attributes that may be used with tc:conditional.

Tag Attributes of tc:conditional

Tag attribute	Attribute value	Description

`compare`	`"Detail[TreePosition]/DetailValue[FieldName],Value"`
	or
	`"System[SubObject],Value"`
	Note: *Do not type a space before or after the forward slash (/) or comma.*
		This tag attribute compares the value from the record with the value specified in the tag. If the value in the record is the same as the value you specify here in the tag, then the condition is met.
		Custom fields:
		1. Replace TreePosition with the actual category tree position of the custom field.
		2. Replace FieldName with the name of the desired custom field.
		3. Replace Value with the value that is expected for the condition to be met.
		4. For more details, see Using tc:conditional to Compare Custom Field Values.
		Sub-objects:
		1. Replace SubObject with the name of the sub-object. Do not type List at the end of the name of the sub-object (AddressList, etc.). For example:
		`Address, Assignee, Email, Fax, InetAddress, Phase, Phone, Relation, Resource, Skill`
		2. Replace Value with the Name of the sub-object type that you want to look for.
		For more details, see Using tc:conditional to Compare Sub-objects.

`test`	`"Detail[UCOD_TREE]"`
	(testing for a category)
	or
	`"ObjectAttribute"`
		You can use this tag attribute for checking whether a value exists in system fields only (including categories). The test tag attribute cannot be used for custom fields.
		Testing for a category:
		Use the test attribute with this syntax to test whether a category exists (has been added) in the current object record. Replace UCOD_TREE with the tree position of the category you want to test.

		For example, if you want to test whether the Office (OFFI) category exists for Expense, replace UCOD_TREE with EXPE_OFFI. **Testing for an object attribute value:** Use this tag attribute to test whether a value exists for the object attribute you specify. If it exists, then any content within tc:conditional is generated. You can use this tag attribute to test: • Whether a value exists for almost any type of object attribute. Types of data you can test include: Date, Text, Number, Object, Memo Text, and so on. ***Note:*** *You cannot use this tag attribute to test a boolean value such as a check-box.* • Whether there are any instances of a sub-object (for example, whether there is at least one Assignee added to a record) • Whether a category has been added to a record. ***Important:*** *When you are using test to see if there is a value and that value is of the data type object (such as a parent record), the tc:conditional tag actually navigates to that object. This means that if you nest any tags within tc:conditional, you must write them relative to the object that you have navigated to.*

`condition`	`"greater"`
	`"less"`
	`"equal"` (default)
		You can use this attribute in conjunction with the compare attribute when setting a tc:conditional tag for a number or date field.
		
		• Use greater to indicate that the condition will be met when the number or date value from the system is greater than the value specified in the compare attribute.
		• Use less to indicate that the condition will be met when the number or date value from the system is less than the value specified in the compare attribute.
		
		The default value for this attribute is equal. When you do not include the condition attribute in the tag, it is assumed that the Document Generator should consider the condition to be met when the value in the system is equal to the value you specify in the compare attribute value.

negate	"yes" or "no" (default)	This attribute reverses the logic of tc:conditional. You can use negate with either compare or test. When you add this attribute to the tag with the attribute value equal to "yes," then the Document Generator will only generate what is nested within tc:conditional when the condition is NOT met.

1.1.19.2.8.1 Using tc:conditional to Compare Custom Field Values

The compare attribute of tc:conditional is useful for comparing a custom field to a value that you specify in the tag. If the custom field has the value that you specify, then the condition is found to be true, and the content nested within the tc:conditional tag is generated.

You can use compare with the following types of custom fields. For each type of custom field, you must provide the value of the compare tag attribute in the specified format.

Tag Attribute Value Formats for Comparing Custom Fields

Custom field type	Format of value	Example tag
Check-box	true or false	<tc:conditional compare="Detail[CLAM]/DetailValue[PoliceInvolved],true">
Number	Value must be typed as an integer (without decimals or commas)	<tc:conditional compare="Detail[CLAM_AUTO]/DetailValue[ClaimEstimate],10000" condition="greater">
Text	Exact text that you want to compare	<tc:conditional compare="Detail[CLAM]/DetailValue[LossLocation],Near Residence">
List	The name of the lookup table item.	<tc:conditional compare="Detail[CLAM]/DetailValue[Fatality],Yes"> ***Caution:*** *If the lookup table item name has a comma in it, then this tag will not function properly.*
Date	YYYY-MM-DD	<tc:conditional compare="Detail[CLAM]/DetailValue[AwardDate],2003-02-26">

Example:

```
<tc:conditional compare="Detail[CLAM]/DetailValue[DrunkDriver],true">
   <tc:file select="Statutes/DrunkDriverStatute.xml" />
</tc:conditional>
<tc:conditional compare="Detail[CLAM]/DetailValue[Speeding],true">
   <tc:file select="Statutes/SpeedingStatute.xml" />
</tc:conditional>
```

In this example, the custom fields **DrunkDriver** and **Speeding** are check-boxes within the project Claim. The values specified for each is true, meaning that if a check-box is checked, then the XML file specified for that field will be included in the generated document. For example, if the **DrunkDriver** check-box is checked, the XML file **DrunkDriverStatute** is included in the generated document since the true condition that you specified was met.

1.1.19.2.8.2 Using tc:conditional to Compare Sub-objects

The compare attribute of tc:conditional can also be used to check whether an instance of a certain type of sub-object exists. If the type of sub-object that you specify in the tag has been added to the record, then the condition is found to be true, and the content nested within the tc:conditional tag is generated.

You can use the compare attribute to check any sub-object that has a Type object attribute. For example, Address, Assignee, Phone, and Email all have a Type.

Example:

```
<tc:conditional compare="System[Assignee],Adjuster">Please contact
your claim adjuster for more information.</tc:conditional>
```

In this example, if the Assignee List in the record contains an assignee of the type Adjuster, then the text inside the tc:conditional tag is generated. **Adjuster** is an example of the type of assignee.

1.1.19.2.8.3 Using tc:conditional to Test for Object Attribute Values

The test attribute of tc:conditional is used to test whether a condition is true. You can test any system fields (object attributes) belonging to an object, EXCEPT for check-boxes. A condition is true when a value exists for that attribute. For example, you can test whether a custom object record such as Claim has a parent selected in the **Parent** system field.

You can also test the list of sub-objects added to a record to check whether any instances of that sub-object exist. If at least one instance of the sub-object is found, then the condition is true, and the content nested within the tc:conditional tag is generated. For example, if there is at least one Address added to a Contact, then the condition is true.

Unlike the compare tag attribute, the test attribute does not check the type of the sub-object. Any instance found proves the condition to be true.

Example:

```
<tc:conditional test="MainAssignee">
Please contact the following associate with any questions you might
have about your claim.
   <tc:data select="User">
   <tc:data select="Contact">
   <tc:data select="FirstName" />
```

```
    <tc:data select="Name" />
    </tc:data></tc:data>
</tc:conditional>
```

In this example, the text and tags that are nested within tc:conditional will only be inserted into the generated document if a Main Assignee exists for the Claim record.

1.1.19.2.8.4 Testing for a Category

You can use the test tag attribute to test whether a certain category has been added to a record. This is a useful way to include content in a generated document only when the main object record has a certain category added to it.

Example:

```
<tc:conditional test="Detail[CLAM_HOME]">
We have the following information about your loss in our records:
    Caused by natural disaster:
    <tc:detail select="CLAM_HOME/NaturalDisaster" />
    Date of Loss:
    <tc:detail select="CLAM_HOME/LossDate" format="" />
    Summary of Loss from your written statement:
    <tc:detail select="CLAM_HOME/LossSummary" />
    Estimated Dollar Amount of Loss:
    <tc:detail select="CLAM_HOME/LossEstimate"
    numberFormat="#,##0.00" />
</tc:conditional>
```

In this example, the text and custom fields contained within the tc:conditional tag will only be inserted into the generated document if the category CLAM_HOME has been added (exists) in the Claim record.

1.1.19.2.9 tc:block

The tc:block tag is used to group tc:conditional tags. The tc:conditional tags on their own can each have nested content that is only included in the generated document when the condition is true. However, when nested within a tc:block tag, multiple tc:conditional tags are handled differently.

The advantage of tc:block is to generate the same content, such as retrieving another XML file, when at least one of a group of conditions are met. For example, if one or more of five check-boxes is checked, then retrieve an XML file. Each tc:conditional tag that is nested within tc:block must have exactly the same content nested within it, so that no matter which condition is found to be true, the Document Generator generates the same content.

When tc:conditional tags are nested within tc:block, the Document Generator reads through each tc:conditional tag and checks the condition. As soon as it encounters a tc:conditional tag with a true condition, it generates that tag's nested content. It then ignores the rest of the tc:conditional tags in the block.

The following points will help you use tc:block:

- If you have several tc:conditional tags to include in a document template but they have different content that they will generate, then do NOT nest them within a tc:block tag. The purpose of using tc:block is to allow the same content to be generated, no matter which condition is met.
- You cannot nest any tags within a tc:block tag except for tc:conditional tags. Other tags can be nested within those tc:conditional tags, but they cannot be nested directly within the tc:block tag.
- Like tc:conditional, you can nest tc:block as needed within other tags.

Example:

```
<tc:block>
    <tc:conditional compare="Detail[CLAM]/
    DetailValue[DrunkDriver],true">
        <tc:file select="Statutes/RecklessStatute.xml" />
    </tc:conditional>
    <tc:conditional compare="Detail[CLAM]/DetailValue[Speeding],true">
        <tc:file select="Statutes/RecklessStatute.xml" />
    </tc:conditional>
</tc:block>
```

In this example, the custom fields DrunkDriver and Speeding are specified. The conditions for these check-boxes are each set to true.

Note: *The tag nested within each tc:conditional tag is exactly the same. If only one of these fields is checked, then the XML file named RecklessStatute.xml will be retrieved and included in the generated document.*

1.1.19.2.10 tc:compute

The tc:compute tag can add, subtract, multiply, or divide two numeric values, depending on the mathematical operator specified.

When using tc:compute, you must know the following concepts:

- Each value can be either a system field, custom field, or value specified within the template, but they must both be numerical values. Otherwise, TeamConnect will display an error and the document will not generate properly.
- You can nest tc:compute as needed within other tags.
- In order to specify which two values to compute, you nest tags within tc:compute that point to numerical values.
- The tc:compute tag can be described as "Field1 (+, -, *, /) Field2". For example, for subtracting, the second nested field is subtracted from the first nested field. For dividing, the first field is divided by the second field.

The following table describes the tag attributes for tc:compute.

Usage of tc:compute

Tag attribute	Attribute value	Description
method	"Operator"	Enter one of the following to specify the mathematical operation between the specified number fields: "+" for addition "-" for subtraction "*" for multiplication "/" for division
format	"format"	Specify the desired number format that will be displayed in the generated document. For a description of possible attribute values, see the description for numberFormat in the Optional Tag Attributes for tc:data and tc:detail table.

Example:

```
<tc:compute method="-" format="###0.00">
    <tc:detail select="CLAM/ClaimEstimate" />
    <tc:detail select="CLAM/ClaimPaid" />
</tc:compute>
```

In the example given, the compute method specified is subtraction "-". CLAM/ClaimEstimate and CLAM/ClaimPaid are number fields for the project Claim. Since CLAM/ClaimEstimate is specified first, CLAM/ClaimPaid will be subtracted from CLAM/ClaimEstimate to give the resulting value in the generated document.

To compute a value from one TeamConnect system or custom field and a value specified within the template, use tc:operand in conjunction with tc:compute. See tc:operand for details.

1.1.19.2.11 tc:operand

The tc:operand tag can be nested within tc:compute to specify a static value to compute with a value from a number field.

Example:

```
<tc:compute method="*" format="###0.00">
    <tc:detail select="CLAM/ClaimEstimate" />
    <tc:operand value=".2" />
</tc:compute>
```

This example calculates a value by multiplying the value in the Claim Estimate field by .2. The purpose of this example is to find a percentage of a value in TeamConnect and insert this percentage value into a generated document.

1.1.19.2.12 tc:search

The tc:search tag is used to automatically retrieve data from related objects of the current object. When you want to retrieve data from related objects automatically, rather than having the user select the related objects from which data should be retrieved, use tc:search.

For example, if you are creating a document template for Policy and you want to automatically retrieve data from its child Claim records, you would use tc:search to find the desired Claim records and get data from them.

You can retrieve data from the following related objects by using tc:search to navigate to them:

- A system object that is directly related to the current object, such as its involved or history.

  Note: *Documents are not directly related to an object. Therefore, you will not be able to retrieve information or content about a document using tc:search. You will need to use tc:filter instead. For information, see Filters for Non-Related Objects.*

- A child custom object of the current custom object.
- An embedded object of the current custom object.
- Child accounts of the current account.

Points To Remember

When using tc:search, you must know the following points:

- Unlike tc:filter, when you use tc:search you cannot guarantee that data will be retrieved from only one record. The Document Generator is "searching" for records according to your tag, similar to a search screen in the user interface.
- If you use tc:search without any tc:conditional tags, then the data will be retrieved from all instances of a related object and included in the generated document. For example, you may want to include the all of the claims filed against a particular policy.
- By nesting the tc:conditional tag within tc:search, you can specify a criterion for the related object records from which to automatically retrieve data.
- You can nest tc:search within other tags as needed to navigate to the appropriate object. For example, after navigating to the Contact object through the contact-centric field in a custom object (by using tc:data), you can use tc:search to get that contact's History records.

When NOT to Use tc:search

You must not use tc:search when:

- You want the user to select which related object record(s) from which data will be retrieved. Instead, use tc:filter.
- You want to retrieve data from a record that is not related to the main object record in any way. Instead, use tc:filter so that the user can select the record.
- You want to retrieve data or content from a document record. Instead, use tc:filter to select a non-related record.

- You want to retrieve data from sub-objects. Instead, use tc:loop (to automatically identify the sub-objects) or tc:filter (to have the user select the sub-objects).

The following table describes the tag attributes for tc:search.

tc:search Tag Attributes

Tag attribute	Attribute value	Description

`link`	`"ObjectName"`	Type the name of the object to which you want the Document Generator to navigate from the current object. The name of the object should be typed as shown:
		`Appointment`
		`Account`
		`Expense`
		`History`
		`Involved`
		`Task`
		Project (for child custom objects and embedded objects only)
		Note: *You must include the qualifier attribute, if you are using tc:search to retrieve child objects.*

`isForChild`	`"yes"` or `"no"`	With a value of "yes," this attribute tells the Document Generator that you are retrieving records that belong to one of the following: • A child object of the current object. • An embedded object of the current object. Incidentally, this means that the object you are navigating to resides in the same (current) table in the database. By default, "no" is the value for this tag attribute. Make sure to use "no" when you are identifying an object that is not a child of the current object.
`qualifier`	`"Application_ProjObjectDefinition_UniqueCode=UCOD"`	This tag attribute is required when you are using tc:search for a child custom object, child account, or an embedded object. This attribute value identifies the child object using the following values: 1. Path of the object attribute that uniquely identifies the child or embedded object, which is Application_ProjObjectDefinition_UniqueCode.

		2. Unique code of the child or embedded object, which can be copied from the Object Definition List in the user interface.

The following example illustrates how tc:search can be used.

Example 1:

```
<tc:search link="Project" isForChild="yes"
qualifier="Application_ProjObjectDefinition_UniqueCode=CLAM">
Claim Information (Child of Policy)
    Claim Name: <tc:data select="Name" />
    Claim Number: <tc:data select="NumberString" />
</tc:search>
```

In the example above, tc:search is retrieving only child object records for Claim, since Policy may have other child custom objects besides Claim.

Rather than including data from all related object records that are identified by tc:search, you may want to include only data from those records that meet certain criteria. For example, instead of the names of all the Involved Contacts added to a policy, you may want to include only the names of Involved Contacts that have a certain category, such as Additional Driver.

Example 2:

```
<tc:data select="Parent">
Parent (Policy) Information
    Number: <tc:data select="NumberString" />
    Name: <tc:data select="Name" />
    Policy Holder: <tc:data select="Contact"><tc:data
    select="FirstName" /> <tc:data select="Name" /></tc:data>
    Additional Drivers: <tc:search link="Involved"><tc:conditional
    test="Detail[PLIV_ADDR]"><tc:data select="Contact"><tc:data
    select="FirstName" />
    <tc:data select="Name" />
    </tc:data></tc:conditional></tc:search>
</tc:data>
```

In this example, the document template is for the Claim object. The template retrieves data from the Involved of the parent of Claim, which is Policy. First, the template navigates to the parent object. Once the template is pointing at Policy, tc:search is used to identify the Involved records. The tc:conditional tag is then used to retrieve data only from those Involved records with the Additional Driver category (role) added to them.

1.1.19.2.13 tc:filter

You can define a filter for the document template so that when the user launches the Document Generator, a filter page opens. The user can then select a record, such as a claimant (an Involved), from a drop-down list. Data from this record can then be retrieved in the content of the document.

In the content section of a document template, tc:filter is used to switch focus from the current object record to the record that will be selected by the user in a filter page. Once this object is selected as the current object, you can retrieve data from it using other tags.

For example, if your document template is for a custom object named Claim, you may use tc:filter to retrieve data from involved, appointment, task, history, child objects, or any other objects that are related to Claim and which have been selected by the user in filter pages.

In the header section, you will need to specify how the filter page will appear in the end-user interface of the Document Generator. You will learn more about this in Defining Filter Pages.

When to Use tc:filter

Use tc:filter in the content section when you want to retrieve data from any of the following records or sub-objects which have been selected by the user in filter pages:

- A related object. The related object can be any of the following:
  - Child custom objects
  - Embedded objects
  - Child accounts (if the main object is an account)
  - Related system objects (for example, a project's involved, appointment, or history records), except document.
- An object that is not directly related to the main object.

  For example, the main object of your **template** is Policy, whose child object is Claim. You may want the user to select the involved parties of the related Claims.
- A sub-object.

  For example, you may want the user to select an Address, Assignee, or Email Address. You can define filter pages that enable users to select these sub-objects.
- An object that is not related to the main or current object, such as Documents.

  For example, you can use tc:filter to retrieve data from a Contact that the user selects, when that Contact is not related to the main object record.

When NOT to Use tc:filter

You would not use tc:filter in the content section under the following circumstances:

- When you do not want the user to select the related records or sub-objects. If this is the case, use tc:search or tc:loop to automatically identify the records and retrieve the correct data.
- When the relationship from the main object to the desired object is found in a custom field.

  For example, if Party and Claim are custom objects and there is a custom field in Claim of type Custom Object that is used for selecting a Party record, then you would navigate to the Party record using tc:detail.
- When the relationship from the main object to the desired object is directly found in a system field.

  For example, from a Claim record, its parent Policy record is found in the system field Parent. Therefore, tc:data is the tag used to get to the data in the Policy record.

You would use tc:filter, however, if you want the user to select an object that is related to the record selected in the system field. For example, if you wanted to use records selected by the user from the Involved records of the Policy mentioned above, this would require the use of tc:filter.

- When you want to retrieve data from a sub-object and you do not want the user to select an instance of the sub-object.

  For example, if you want to navigate to the Contact record of Assignees and automatically list the names of all assignees from their Contact records (instead of having the user select an Assignee), then you would use tc:loop.

Points To Remember

When using tc:filter, you must know the following concepts:

- The tc:filter tag does not actually retrieve data and insert it in the generated document. tc:filter is used for navigating to the correct object in TeamConnect to identify the data you need. In order to retrieve data from that object, you use other Document Generator tags, such as tc:data and tc:detail.
- You can nest other tags within tc:filter as needed in order to navigate to the desired objects.
- You must always define the filter using a tc:filter tag in the header of the document template. In the content section, you use tc:filter again to reference the filter you defined that identifies the record(s) from which you want to retrieve data. You will learn more about this in Defining Filter Pages.
- The filter that you define in the header section of the document template can allow the user to select one or multiple related object records.
- Use tc:filter when you need to retrieve data from a sub-object that is selected by the user. Then, in the header section, use the tc:filter tag to define that filter as a filter page.
- You can refer to the following for samples of document templates using tc:filter:
  - Automatically Retrieving Data from Related Objects.
  - Including Information When Certain Conditions Are Met.

The following table describes the attributes of the tc:filter tag when you are using it in the content section of the document template.

tc:filter Tag Attribute

Required?	Tag attribute	Attribute value	Description
Yes	`select`	`"FilterName"`	Type the unique name for the filter that will appear in the filter page in the end-user interface of the Document Generator. This is the name that you define for the filter in the header of your document template.

			Tip: *If you have not yet defined the filter in the header section of the document template, you can choose a name for it now and use this name later when you define the filter. For more details, see Defining Filter Pages.* ***Important:*** *The tc:filter tag does not retrieve any data by itself; you must nest other tags within the tc:filter tag to retrieve data.*
No	`list`	`"yes"` or `"no"`	By default, this attribute is set to "no". You must set list="yes" when the filter you are using is a hidden filter. For details, see Defining Filter Pages.

Example:

```
<tc:filter select="InvolvedParty">
    <tc:data select="Contact">
    <tc:data select="FirstName" />
    <tc:data select="Name" />
    </tc:data>
</tc:filter>
```

In the example given, the data retrieved by TeamConnect will be the first and last name of the Contact, who is an Involved Party of the Claim record. The user will choose which Involved Party's data should be retrieved when invoking the Document Generator. InvolvedParty is the filter chosen. The filter that you select here is defined in the header section of the document template.

You can find more examples using tc:filter in Document Template Samples.

1.1.19.2.14 tc:input

The tc:input tag retrieves text that is entered by the user who invokes the Document Generator. The user will enter text within fields that you will create by defining them in the header section of the document template. The information retrieved by the tc:input tag is not data within any TeamConnect records.

When using the tc:input tag in the content section of the document template, you reference a corresponding tc:input tag in the header section of the document template where you define the input page.

You can nest tc:input where needed in the content section. It is not affected by the outer tags where it is nested.

The following table indicates how to use tc:input in the content section to retrieve the selection made by the user in the input page.

Usage of tc:input

Tag attribute	Attribute value	Description
select	"InputName" or "InputForm.InputName"	Type the unique name for the text field in which the user will enter information in an input page. The name that you type here in the content section of the document template must correspond to an input field that you define in the header. If the field is defined as one of multiple fields in one input page, then the InputName must reflect which input page contains the field, as shown.

Example: `<tc:input select="UserInitials" />`

In the example given, UserInitials is the input field chosen. When the document is generated, the text that the user entered in this field in the input page will be inserted in place of this tag.

The input field is defined in the document template's header section. For details about defining input pages, see Defining Pages for User Input. See also the example letter in the Document Generator Template Page Two image for examples of using tc:input.

1.1.19.3 Nesting to Navigate

To retrieve the data you need in a document template, you must utilize the built-in relationships among the objects in the Object Model. These relationships are reflected in the tags you write in the document template as either nested tags or paths written within a tag attribute.

Nesting tags in a document template is equivalent to navigating throughout the Object Model to build paths to the data you want to retrieve. Each time you nest a tag, you are "traversing" between two objects in the Object Model using a relationship that exists between them. When you open a tag but do not close it, you point the Document Generator to the object to which that particular item has a relationship. The next tag you use refers to that object. When you are finished using data and relationships from that object, close the nested tag to return to the object that you started from.

Nesting to navigate can start with any piece of data that identifies another object. System fields, custom fields, related objects, and sub-objects can all have relationships to other objects. Therefore, you can even nest a tag after pointing to a system field or a custom field.

In order to find out whether a field points to another object, or how two objects are related, see Object Model: Read This First. Information here indicates whether an object attribute points to another object by indicating the data type Object in the Data Type column of each table.

For example, suppose you have two custom objects named Claim and Policy, where:

- Claim is a child of Policy.
- Policy is contact-centric.
- Claim has a custom field of type Custom Object that is used for selecting another claim as the Master Claim.

If the following tags were in a document template for the Claim object, they would retrieve data from the policy that is the parent of the master claim:

```
<tc:detail select="CLAM/MasterClaims">
    <tc:data select="Parent">
    <tc:data select="Contact">
    <tc:data select="FirstName" />
    <tc:data select="Name" />
    <tc:loop select="AddressList" qualifier="Type_TreePosition=ADDR_BUS1">
    <tc:data select="Street" /> <tc:data select="City" />
    <tc:data select="State" /> <tc:data select="PostalCode" />
    </tc:loop>
    <tc:data select="SsOrTaxNumberString" />
    <tc:data select="DriverLicense" />
    </tc:data>
    </tc:data>
</tc:detail>
```

The following table provides a detailed explanation of each line of code in this example.

Line #	Example	Description
1	`<tc:detail select="CLAM/ MasterClaims">`	This tc:detail tag points to the custom field MasterClaims in the current claim. • Since the tag is not closed, the Document Generator navigates to the Claim record selected in this custom field, known as the master claim. • Also, since the tag is not closed, nothing is printed in the generated document.
2	`<tc:data select="Parent">`	This tc:data tag points to the system field where the parent record is selected. • Since the tag is not closed, the Document Generator navigates to the parent record, which is a policy. • Also, since the tag is not closed, nothing is printed in the generated document.
3	`<tc:data select="Contact">`	This tc:data tag points to the system field where a contact is selected for a contact-centric record.

		• Since the tag is not closed, the Document Generator navigates to the Contact record. • Also, since the tag is not closed, nothing is printed in the generated document.
4	`<tc:data select="FirstName" />`	This tc:data tag points to the system field where the contact's first name is stored. • The tag is closed, so the Document Generator does not navigate but remains in the Contact object. • Also, since the tag is closed, the value of the field is printed in the generated document.
5	`<tc:data select="Name" />`	This tc:data tag points to the system field where the contact's last name is stored. • The tag is closed, so the Document Generator does not navigate but remains in the Contact object. • Also, since the tag is closed, the value of the field is printed in the generated document.
6	`<tc:loop select="AddressList" qualifier="Type_TreePosition=ADDR_BUS1">`	This tc:loop tag points to the Address sub-object of the Contact object. • The qualifier attribute limits the sub-object to addresses of type BUS1 only. • Since the tag is not closed, the Document Generator navigates to the list of addresses added to the contact record. • Also, since the tag is not closed, nothing is printed in the generated document.

7	`<tc:data select="Street" />`
	`<tc:data select="City" />`
	`<tc:data select="State" />`
	`<tc:data select="PostalCode" />`
		These tc:data tags point to the system field where the street, city, state and postal code are stored for the address.
		• Each tag is closed, so the Document Generator does not navigate but remains in the Address sub-object.

		• Also, since each tag is closed, the value of the field is printed in the generated document.
8	`</tc:loop>`	This is the closing tag for tc:loop. When this tc:loop is closed, the Document Generator navigates back from the Address sub-object to the Contact object.
9	`<tc:data select="SsOrTaxNumberString" />` `<tc:data select="DriverLicense" />`	These tc:data tags point to the system fields where the contact's social security number and driver license number are stored. • Each tag is closed, so the Document Generator does not navigate but remains in the Contact object. • Also, since each tag is closed, the value of the field is printed in the generated document.
10	`</tc:data>`	This is the closing tag for the tc:data tag that we left open on line **3**. When this tc:data tag is closed, the Document Generator navigates back from the Contact to the Policy object.
11	`</tc:data>`	This is the closing tag for the tc:data tag that we left open on line **2**. When this tc:data tag is closed, the Document Generator navigates back from the Policy object to the Claim object.
12	`</tc:detail>`	This is the closing tag for the tc:detail tag that we left open on line **1**. When this tc:detail tag is closed, the Document Generator navigates back from the Master Claim selected in the custom field to the starting Claim.

1.1.19.4 Locating Object Attribute Names

As you construct tags in your document template, you will identify items using the select tag attribute. The name of the item that you identify in a tag directly corresponds to the object attributes in the TeamConnect Object Model. Each object's attributes are listed in Object Model: Read This First and in the additional tables it points to.

When you identify an object attribute in a tag attribute, you must type the name of the object attribute correctly in the tag. For example, the first letter of the object attribute must be upper case when you type the object attribute name in a document template tag.

The following procedure will help you find and type the object attribute names correctly as you construct tags in your document template.

To find the correct object attribute for a tag

1. Open an online version of this document in Adobe Acrobat®.
2. In the Contents pane on the left, find Object Model: Read This First. Additional related collections of reference tables follow this entry.
3. Press the **V** key on your keyboard to turn on Acrobat's Select Text Tool, or click the Select Text Tool icon on the toolbar.
4. In the Bookmarks pane on the left, find and click the name of the table for the current object you are working with. For example, if the latest tag in your document template is pointing to a project record, go to the TProject table.
5. In the object table, find the attribute you need.

   Tip: *Read the* ***Comments****,* ***Field in User Interface****, and other columns to help identify the correct object attribute.*

6. Double-click the name of the attribute in the **TeamConnect Attribute** column to select it.
7. Open your document template and place your cursor where you want to include a reference to an object attribute in a tag.
8. Use the **Paste Special** command to paste the name of the object attribute as **Unformatted Text**.
9. Change the first letter of the object attribute to upper case.

   Important: *The Document Generator will not recognize the tag attribute value unless it is capitalized.*

10. If the object attribute has an arrow after it, delete the arrow or whatever character gets pasted in place of the arrow.
11. Delete any extra spaces before or after the attribute name.
12. Finish the tag as needed.

1.1.19.5 Inserting Tags and Preparing RTF Files

Based on the previous sections, you should be able to insert the correct document template tags to retrieve data. You will need to replace all of your placeholders with the correct tags.

After inserting all of the tags into your document template, your file must be converted to RTF, if you have not already been saving it as an RTF file.

To insert tags and prepare the RTF file

1. With your word processing application, open the document template text that you previously created.
2. Replace all of the placeholders with their corresponding document template tags. For more information about how to identify the data that you need from the Object Model, see Locating Object Attribute Names.
3. Format the text and the document template tags exactly as you would like the information to appear in the generated document (specifically, enter all line returns, indentations, line formatting, font size, and so on).

   The Document Generator Template Page One image is an example of the text from the Document Generator Template - Text and Placeholders image that has been replaced with corresponding Document Generator tags. Notice that the formatting (line returns, indentation, spacing, tables, and so on) used in the Document Generator Template Page One image is reflected in the generated document shown in the Document Generator Template - Text and Placeholders image.
4. Save the content with the extension RTF and close your word processing application.
5. Find the RTF file you just saved and open it with a text editor such as Microsoft Notepad™.

   Caution: *Do not use Microsoft WordPad™ to edit your RTF file. You must be able to see all formatting as text in your text editor.*

6. After opening an RTF file that was created with Microsoft Word™, all double quotes (") that were entered in Word may be displayed as \'94 in the text editor. Similarly, all single quotes (`) may be displayed as \'92.
   a. Find all instances of \'94 that appear within the source code and replace with double quotes (").
   b. Find all instances of \'92 that appear within the source code and replace with single quotes (`).
7. Save the file.

This completes the content section of the document template. See the Document Generator Template Page One image for an example of a content section with Document Generator tags.

You must now build the header section and document tags of the template around this source code, as described in Using Document Template Headers and Completing the Document Template.

<tc:data select="Name" />
<tc:data select="NumberString" />

<tc:date format="EEE, MMM d, yy"/>

<tc:filter select="SelectInvolvedContact"><tc:data select="FirstName" /> <tc:data select="Name" /></tc:filter>
<tc:filter select="InvolvedContactAddress"><tc:data select="Street" />
<tc:data select="City" />, <tc:data select="State" /> <tc:data select="PostalCode" /></tc:filter>

<tc:filter select=" SelectInvolvedContact "><tc:data select=" Salutation" />, </tc:filter>

ClientGuard was informed of your recent auto accident that occurred on <tc:detail select="CLAM/LossDate" />. We are currently processing this claim and according to our records, the following individuals have filed claims regarding your auto accident:
<tc:filter select="involvedinaccident" list="yes"><tc:data select="Contact/FirstName" /> <tc:data select="Contact/Name" /></tc:filter>

The following passengers in your vehicle are claiming injury as a result of this accident:
<tc:filter select="passenger" qualifier="IVPT_DEFA_INPS" list="yes">
<tc:data select="Contact/FirstName" /> <tc:data select="Contact/Name" /></tc:filter>

From the police report, we have determined that you were charged with the following:
<tc:detail select="CLAM/DrunkDriver" />
<tc:detail select="CLAM/Speeding" />
<tc:detail select="CLAM/RecklessDriving" />
<tc:detail select="CLAM/LoudMusic" />

For the charges listed above, you were in violation of the following California statutes:
<tc:conditional compare="Detail[CLAM]/DetailValue [DrunkDriver],true">
<tc:file select="Statutes/DrunkStatute.xml" /></tc:conditional>
<tc:conditional compare="Detail[CLAM]/DetailValue [Speeding],true">
<tc:file select="Statutes/SpeedingStatute.xml" /></tc:conditional>
<tc:conditional compare="Detail[CLAM]/DetailValue [LoudMusic],true">
<tc:file select="Statutes/LoudMusicStatute.xml" />
</tc:conditional>

In addition, based on the violations listed above, you were also guilty of breaking the following statutes:
<tc:block>
<tc:conditional compare="Detail[CLAM]/DetailValue [DrunkDriver],true">
<tc:file select="Statutes/RecklessStatute.xml" /></tc:conditional>
<tc:conditional compare="Detail[CLAM]/DetailValue [Speeding],true">
<tc:file select="Statutes/RecklessStatute.xml" /></tc:conditional>
<tc:conditional compare="Detail[CLAM]/DetailValue [LoudMusic],true">
<tc:file select="Statutes/RecklessStatute.xml" />
</tc:conditional>
</tc:block>

The balance due for this claim and the services provided by ClientGuard is a total of <tc:compute method="-" format="#####.##"><tc:data select="ClaimEstimate" /><tc:data select="ClaimPaid" /></tc:compute> dollars. Please send your payment to ClientGuard as soon as possible.

<tc:input select="UserInitials" />
<tc:input select="LetterDate" />

Document Generator Template - Text and Tags - Page One

<tc:data select="Name" />
<tc:data select="NumberString" />

If you have any questions regarding your remaining balance, please contact any of the following claim associates here at ClientGuard: <tc:loop select="AssigneeList">

<tc:data select="User/Contact/FirstName" /> <tc:data select="User/Contact/Name" />	<tc:data select="Type/Name" />	<tc:data select="User/Contact/DefaultPhone/PhoneString" />

</tc:loop>

If you have any additional information you would like to include with your claim, please contact <tc:loop select="AssigneeList" qualifier="Type_TreePosition=CLAM_SPRV"> <tc:data select="User/Contact/FirstName" /> <tc:data select="User/Contact/Name" /> <tc:data select="User/Contact/DefaultPhone/PhoneString" />.
</tc:loop>

Sincerely,

<tc:user>
<tc:data select="Contact/FirstName" /> <tc:data select="Contact/Name" />
<tc:data select="Contact/Title" />
</tc:user>

<tc:input select="AdditionalComments" />

<tc:input select="UserInitials" />
<tc:input select="LetterDate" />

Document Generator Template - Text and Tags - Page 2

1.1.20 Completing the Document Template

After you have included all of the necessary tags in the content of the document template, you must do the following operations to finish creating the template:

- Create the document tags.
- Create the template header.
- Convert the template to XML.
- Upload the template.
- Test the template.

After you have finished these operations, your template is completed and ready for the end user.

1.1.20.1 Creating Document Tags

The document tag of a document template specifies all the necessary background information such as the name of the template within the Document Generator and the mime type used to open the generated document.

Every document template must have the tc:document tag. One of the final steps you complete as you finalize your document template is adding this tag.

The following table provides a list of the attributes and possible attribute values that must be used with the tc:document tag.

Document Tag Attributes

Tag attribute	Attribute value	Description
`version`	"1.0"	Specify the XML version (currently 1.0).
`mime-type`	"MimeType"	Specify the application that is launched by TeamConnect to open the generated document. The mime-type is for a specific word processor or RTF format. For example, the mime type for Microsoft Word is application/msword, while the mime type for RTF is application/rtf. For mime types msword and rtf, the mime type must match the file type of the template itself. For example, if the mime type is msword, the document template must be a Word (.DOC) file.
`name`	"TemplateName"	Specify the name of the template. This is the name that the user will see when he/she selects templates to generate documents from them.

		• Do not include a file extension in the name. The file extension, such as .doc, is inserted automatically when a document is generated from the template, based on the mime type. • The template name must be unique among templates for the object.
xlmns:tc	"http://www.w3.org/1999/XSL/Transform"	Specifies what name space the prefix "tc" is bound to (used for parsing).

The following is an example of a tc:document opening tag in a document template. The mime type indicates that Microsoft Word is the desired output format and that the name of the template that the user will see is "Notice of Auto Accident."

Document Section

```
<tc:document version="1.0" mime-type="application/msword"
name="Notice of Auto Accident" xmlns:tc="http://
www.w3.org/1999/XSL/Transform">
```

1.1.20.2 Creating Header and Converting Template to XML

Once you have finished inserting tags in the content section of your document template and have identified what filters and input pages you need to define in the header, you are ready to finalize the template by creating the header and converting the file to XML.

Important: *Before completing the following process, verify that you have completed* *Inserting Tags and Preparing RTF Files**.*

To create the header section and convert the template to XML

1. Open the document template, which is an RTF file, in a text editor or HTML editor, such as EditPlus or Notepad™.
2. At the beginning of the document, type the following:

   ```
   <tc:content>
   ```

   Caution: *Do not place a line return after this tag. If the source code for the document content does not start immediately after the tc:content tag, TeamConnect will encounter an error when trying to generate this document.*

3. At the end of the document, type the corresponding end tag:

   ```
   </tc:content>
   ```

   Caution: *Do not place a line return before this tag.*

4. Save the document as a new file with an XML extension.

5. With a text editor such as Notepad or HTML editor such as EditPlus, open the Document Content XML file.
6. At the beginning of the document before the opening tc:content tag, insert the following tags, with a line return after each, in the following order:

   a. `<?xml version="1.0" encoding="UTF-8"?>`

   b. `<tc:document version="1.0" mime-type="MimeType" name="TemplateName" xmlns:tc="http://www.w3.org/1999/XSL/Transform">`

      For more details about the attributes for the document tag, such as the mime type and template name, see Completing the Document Template.

   c. `<tc:header>`
7. Nest the appropriate tc:filter and tc:input tags after the tc:header tag. If your document template does not require any user input, then you will not nest any tags within the tc:header tag.
8. Close the header section by inserting the following tag after the last tc:filter or tc:input tag.

   `</tc:header>`

   Important: *You must include the open and close tc:header tags whether or not you are using tc:filter and tc:input.*
9. At the end of the document after the closing tc:content tag, insert the following tag:

   `</tc:document>`

   This completes the template.
10. Save the XML document.
11. Test your XML document by opening it in Internet Explorer to make sure that your coding has no errors.

After you save the XML document, it is ready to be uploaded into TeamConnect as a document template. For more information, see Uploading Document Templates.

Header Example

The following is an example of a header. This header includes the filter and input tags necessary for the claim notification letter shown in the examples throughout this document. For example, see the document generator template page two image.

```
<tc:header><tc:filter name="SelectInvolvedContact"
searchCondition="Involved_Contact" hidden="no" label="Please select an Involved
Party" displayString="FirstName;Name" displayFormat="* *" />
   <tc:filter name="InvolvedContactAddress" objectFrom="SelectedInvolvedContact"
   test="AddressList" displayString="Type/Name;Street;City" displayFormat="* - *
   *" label="Please Select the Involved's Address."/>
   <tc:filter name="involvedinaccident" searchCondition="Involved"
   hidden="yes" />
   <tc:filter name="passenger" searchCondition="Involved" hidden="yes"
   displayString="DefaultCategory/TreePosition" />
```

```
    <tc:input name="UserInitials" label="Enter your initials here." size="15" />
    <tc:input name="LetterDate" label="Date letter will be sent" size="15" />
    <tc:input name="AdditionalComments" label="Enter any additional comments."
    size="50" maxChars="400" multi="yes" cols="45" rows="2"/>
</tc:header>
```

1.1.20.3 Uploading Document Templates

Now that you have created the content, header, and document sections of your document template and resolved all errors, you can now upload the document template into the Documents area of TeamConnect to make available to users.

To upload document templates into TeamConnect

1. Open the appropriate object definition and go to the object's folder.
2. Open the **Document Templates** folder.

   Important: *Remember that the template must be uploaded to the Custom or System Object on which all of its tags are based. For more information, see Object Dependency.*

3. Click the **Upload File** button on the toolbar. The **Upload New File** window opens.
4. Click the **Browse** button in the **Upload New File** window and find the document template you would like to upload.
5. Click **Upload File**.

   The XML file you created is immediately available to the user when invoking the Document Generator.

The document generator template image is an example of a finished letter that was created with the Document Generator. Compare the finished letter to the template shown in the document generator template page two image. Notice the data that each tag in the document generator template page two image retrieved and inserted into the generated document in the document generator template image.

Caution: *Once you have chosen the directory that will hold your document template, you should not rename, delete, or move that directory. You should also not rename delete or move files within that directory. If you do these operations, the associated documents will no longer work correctly for end users at runtime.*

1.1.20.4 Testing Document Generator Templates

After uploading a document template, you must ensure that it is retrieving data correctly by using it to generate documents with sample data that has been entered into the system. You will need to make sure that the records you use to test your template have the data that is being utilized by the template.

While verifying that your template is functioning as you have intended, use the following checklist to help you identify possible issues. For more information, see Troubleshooting Document Templates for commonly encountered issues.

Keep in mind the following tips and cautions:

- There are no errors when you try to generate the document.
- When you view the generated document, all of the tags retrieve data from the correct sources (for example, user input and TeamConnect data). No tag is skipped unless there is no data for the tag to retrieve.
- When you view the generated document, there are no pieces of tags remaining in the document.
- If the template contains any tc:conditional tags, the content inside the tc:conditional tag is being generated only when the condition is met, and not when the condition is NOT met.
- If the template contains any tc:filter tags, the tags are retrieving data from all related records, according to the qualifier, or according to the user's selection, depending on how you have defined it.
- If the template contains any tc:search tags, the tags are retrieving data from the correct related records.
- If the template contains any tc:loop tags, the tags are retrieving data from all of the correct sub-objects.
- Templates that request fields with null values or empty strings (spaces without characters or numbers) will display in the Missing Data screen.
- If the template contains any tc:conditional tags to test or compare a value that is either null or an empty string, the logic will return false. The Missing Data screen for that document will not display the field for the compared value.
- If the template contains any tc:filter tags, and if there is no selection available that meets the filter criteria, that selection and its associated system and custom fields will not be included in the Missing Data list.

1.1.21 Using EasyDocs

EasyDocs is a method of creating document templates, which has been built upon the Document Generator feature in TeamConnect. EasyDocs allows you to create Document Generator templates without having to manually code the XML tags. Instead, you create an RTF file with mail merge fields and then perform mapping to TeamConnect data in the user interface, using a Data Mapping tool. See Locating the Data Mapping Tool for more information.

EasyDocs helps speed up the process of creating document templates by eliminating much of the potential for human error, because you do not have to manually code the XML tags.

To get started with EasyDocs, review the following explanations:

- EasyDocs vs. Document Generator: Functional Scope
- Creating a Document Template: 5-Step Process Overview
- Required TeamConnect User Group Rights

1.1.21.1 EasyDocs vs. Document Generator: Functional Scope

EasyDocs provides most, but not all, of the functions of Document Generator. If EasyDocs does not provide the data that you need to populate in the generated document, you may need to create document templates using Document Generator XML tags, as described in About Document Generator.

The following table lists the available XML tags in Document Generator and the equivalent merge field type in EasyDocs.

EasyDocs vs. Document Generator

Type of data to retrieve	Document Generator tag	EasyDocs merge field type	Limitations in EasyDocs
System field	tc:data	Basic merge field or automatically mapped merge field	
Custom field	tc:detail	Basic merge field	
Fields automatically retrieved from a sub-object	tc:loop	loop@	EasyDocs cannot retrieve a list of categories added to the record (detailList).
Current date	tc:date	Basic merge field, mapped to today's date	
Current user	tc:user	Basic merge field, mapped to current user	
Content from another file	tc:file	Basic merge field, mapped to file	
Conditional content	tc:conditional	if@	
Fields from a related object retrieved automatically without user interaction	tc:search	loop@	
Fields from a related object, sub-object, or unrelated object, according to user selection	tc:filter	filter@	EasyDocs cannot retrieve data from unrelated objects. Example: A drop-down list of contacts that are unrelated to the current

			record cannot be provided to the end user.
Text entered by user at the time of document generation	tc:input	Basic merge field, mapped to user input	
Several text entry fields in one user input screen	tc:inputForm	Not available in this version	
Conditional content based on multiple conditions	tc:block, used with tc:conditional	Not available in this version	
Computed numeric value	tc:compute	Not available in this version	
Static value used for tc:compute calculation	tc:operand	Not available in this version	

1.1.21.1.1 Creating a Document Template: 5-Step Process Overview

EasyDocs provides a graphical user interface for creating Document Generator templates.

This is how a template is created and configured:

Step 1: Create RTF with Merge Fields.

Step 2: Upload RTF File to TeamConnect.

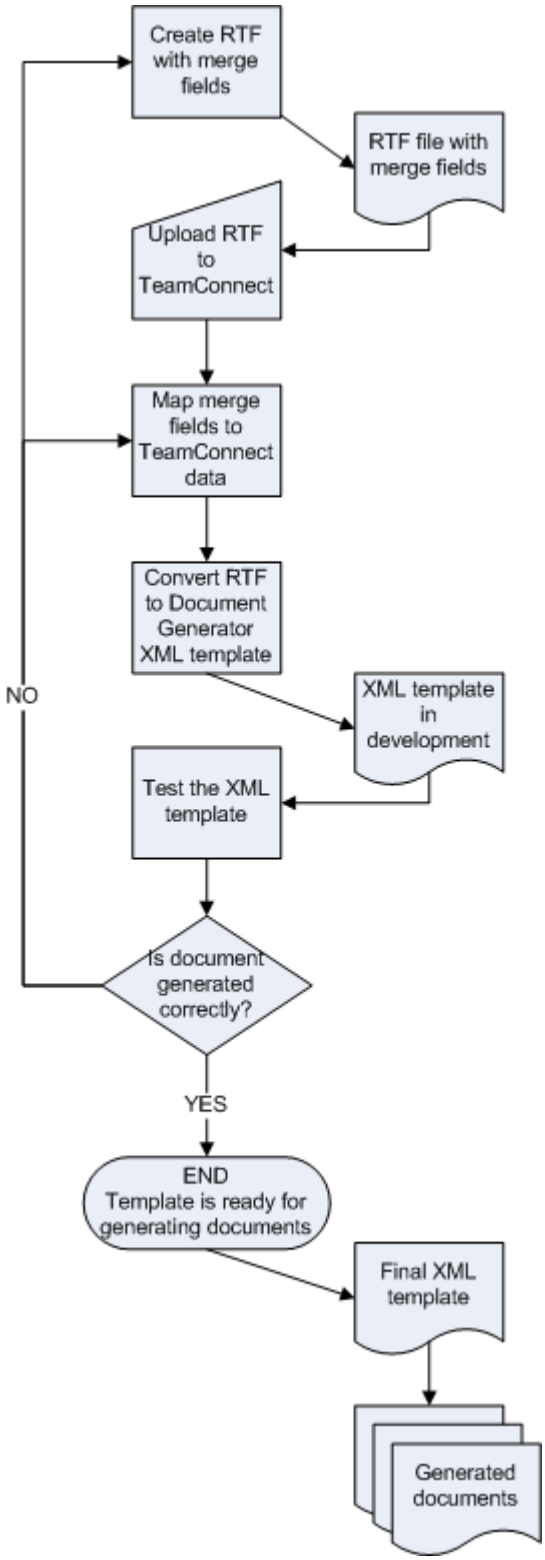
Step 3: Map RTF Merge Fields to TeamConnect Data.

Step 4: Convert RTF File to a Document Generator Template.

Step 5: Test Your Template.

While configuring a template, you will most likely need to cycle through these steps several times until all of the merge fields and mapping are configured properly.

The following flowchart provides a visual depiction of this process.



EasyDocs Template Creation Process

1.1.21.1.2 Required TeamConnect User Group Rights

In order to perform the mapping of an RTF to TeamConnect data and to test a document template, you must belong to a user group that has the appropriate rights assigned to it by an administrator. These include, but are not necessarily limited to:

- For Documents:
  - Read
  - Update
  - Create
  - Perform Template Data Mapping
  - Generate New XML Based Document
- For the record type with which the document template is associated (such as Matters):
  - Read
  - Create
  - Update
  - Also make sure you have rights to its categories and sub-objects (such as assignees).
- For all related records with which the main record type is associated, if data from these records needs to be retrieved in the document (such as involved parties, child or embedded project records, appointments, tasks, and so on):
  - Read
  - Create
  - Update
  - Also make sure you have rights to its categories and sub-objects.

Please ensure that your group has these rights given to it, at a minimum, before proceeding with the instructions in this document.

1.1.21.2 Step 1: Create RTF with Merge Fields

The first step for creating a Document Generator template through EasyDocs is to create a Rich Text Format file (RTF) using Microsoft Word or Corel WordPerfect that includes mail merge fields. The RTF file will become the template in TeamConnect.

Important: *Make sure that you have the appropriate rights assigned to you, as listed in* *Required TeamConnect User Group Rights*.

To create an RTF with merge fields

1. Create a Word or WordPerfect document that you want to convert to a Document Generator template.

2. Insert a separate field of the type "MergeField" for each spot in your document where you would like to retrieve data from TeamConnect.

   For example, if you want a contact name to appear in a letter, you must insert two merge fields, one for the first name and one for the last name, because this is how the name is stored in TeamConnect.

   Be sure to read the tips in step 3 to avoid errors in the fields that you insert.

   Note: *Merge fields appear in your document as field names surrounded by angle brackets, such as <<assignee>>. However, you cannot simply type such names directly into your document. You must use the special menu commands and procedures in Word or WordPerfect to create merge fields.*

   Here is an example of inserting a merge field in Microsoft Word 2002:

   - Position your cursor at the point in your document where you want the merge field to appear.
   - Choose menu items Insert | Field.
   - A dialog appears like that shown in the merge field in Microsoft Word 2002 image. Choose "MergeField" from the list box on the left. Type a "Field name" for your merge field where prompted. Click OK. The merge field appears in the document.

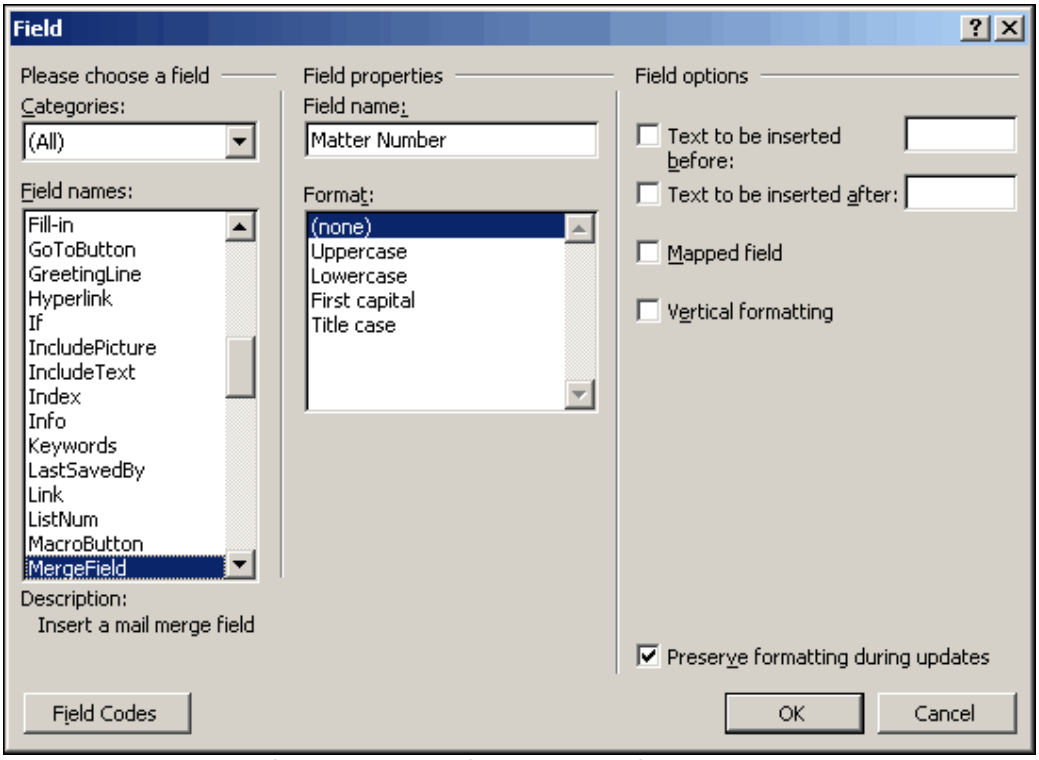
     The simple field names RTF sample image shows an example of an RTF with merge fields that map to TeamConnect data. Most of the merge field names are designed to be easy to read. If you use simple, easy names you must manually use Object Navigator, later, to map each merge field name to its proper data attribute.

     The advanced field names RTF sample image shows a similar RTF, but this one uses precise syntax rules to create more complex names for most of the fields. The advantage of using this special syntax is that the data mapping procedure is done automatically for fields that have names in this syntax. If you already know the relationships between your objects and you know the syntax rules, you can save time by typing field names in this syntax.

     Both RTF files use the special "filter@" and "loop@" tags that will be explained later in this chapter. Both RTF files produce exactly the same output (shown in the sample output from EasyDocs image). The only difference between them is the use of special syntax in some of the field names in the advanced field names RTF sample image.

     For more information about how to set up your merge fields in the RTF, see the following:

     - Basic Merge Fields
     - Automatically Mapped System Fields
     - Automatically Mapped Custom Fields
     - loop@ Merge Fields for Automatically Selecting Related or Sub-objects
     - filter@ Merge Fields for User-selected Related or Sub-objects
     - if@ Merge Fields for Conditional Text



Creating a merge field in Microsoft Word 2002

3. While inserting merge fields, keep the following in mind:
   - Do not insert two merge fields immediately next to each other. You must type a space or other text between merge fields to avoid errors.
   - Merge field names can include these characters: "A-Z", "a-z", "0-9", "." and "_". Do not include spaces in merge field names.
   - Do not include other types of fields in the RTF file that are possible in Word or WordPerfect, such as the current date. TeamConnect only recognizes mail merge fields. To insert the current date, create a basic merge field, which you can later map to the current date.
   - If you use the same merge field name more than once in an RTF to identify the same piece of information, you only have to map it once in TeamConnect.
   - If you need to include multiple merge fields that identify similar data, make sure to give them distinctly different names, because later they will all be listed in a single drop-down list when you are mapping them.
4. Save the RTF file.
5. Upload the RTF file as described in Step 2: Upload RTF File to TeamConnect, after completing your merge fields. If you need to, you can make changes after uploading the file and mapping the merge fields that you created so far.

«Date»

Matter Name: «Matter Name»
Matter Number: «Matter Number»
Main Assignee: «Main Assignee First Name» «Main Assignee Last Name»
Main Assignee Phone: «Main Assignee Phone»

Contact-centric field's contact:
«Contact First Name» «Contact Last Name»
«Contact Address Street»
«Contact Address City», «Contact Address State» «Contact Address Postal Code»

Dear «Contact Prefix» «Contact Last Name»,

The following Involved Parties of type Outside Counsel are involved in this matter:
«filter@Involved Parties»«Involved First Name» «Involved Last Name», «Involved Role»
«filter@»

Here is a list of all Involved Parties, regardless of type:
«loop@Involved Parties»«Involved First Name» «Involved Last Name», «Involved Role»
«loop@»

Sincerely,
«Current User First Name» «Current User Last Name»

«Current User Initials»

Sample RTF File (simple field names)

«Date»

Matter Name: «@name»
Matter Number: «@numberString»
Main Assignee: «@mainAssignee:firstName» «mainAssignee:name»
Main Assignee Phone: «MainAssigneePhone»

Contact-centric field's contact:
«@contact.firstName» «@contact.name»
«@contact.defaultAddress.street»
«@contact.defaultAddress.city», «@contact.defaultAddress.state»
«ContactAddressPostalCode»

Dear «@contact.prefix» «@contact.name»,

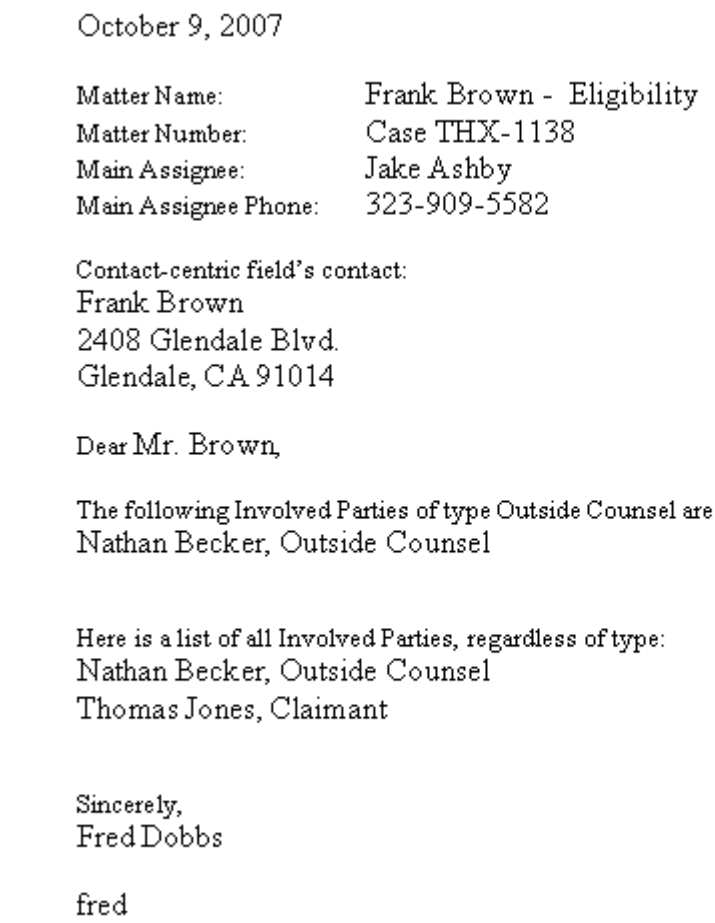
The following Involved Parties of type Outside Counsel are involved in this matter:
«filter@InvolvedParties»«InvolvedFirstName» «InvolvedLastName», «InvolvedRole»
«filter@»

Here is a list of all Involved Parties, regardless of type:
«loop@InvolvedParties»«InvolvedFirstName» «InvolvedLastName», «InvolvedRole»
«loop@»

Sincerely,
«@user:firstName» «@user:name»

«@user.userName»

Sample RTF File (advanced field names)



October 9, 2007

Matter Name: Frank Brown - Eligibility
Matter Number: Case THX-1138
Main Assignee: Jake Ashby
Main Assignee Phone: 323-909-5582

Contact-centric field's contact:
Frank Brown
2408 Glendale Blvd.
Glendale, CA 91014

Dear Mr. Brown,

The following Involved Parties of type Outside Counsel are involved in this matter:
Nathan Becker, Outside Counsel

Here is a list of all Involved Parties, regardless of type:
Nathan Becker, Outside Counsel
Thomas Jones, Claimant

Sincerely,
Fred Dobbs

fred

The sample output from the EasyDocs templates in the previous images

1.1.21.2.1 Basic Merge Fields

The purpose of merge fields in a template is to identify specific elements of TeamConnect data that should be retrieved when a document is being generated.

Basic merge fields are merge fields that do not need to follow any specific naming convention. You can specify any name when you define them in the RTF. Each merge field can be mapped to a simple data element in TeamConnect, such as a number, date, text, or list field.

Basic merge fields can be used to identify each of the following types of information:

- Data from system fields (for example, fields in the **General** block of the record).

  If you want to map system fields automatically, you should follow a naming convention instead. See Automatically Mapped System Fields.

- Data from custom fields (fields that are specifically created for your TeamConnect implementation and belong to specific Categories).

If you want to map custom fields automatically, you should follow a naming convention instead. See Automatically Mapped Custom Fields.

- Text that the user needs to enter manually at the time of generating a document with this template.
- The current date.
- Text from another file.
- Attributes of the current user who is generating the document (usually, the current user's first and last name)

  Also see Predefined Merge Field Codes for predefined mapping to the current user.

For example, if a document template is based on a matter, you may want to retrieve the matter name and matter number. To do this, you can create merge fields with names like **MatterName** and **MatterNumber**, and this will help you to recognize them when you are mapping them in TeamConnect.

The following table provides examples of basic merge fields for a document template that is based on a project (such as Matter). You would need to map each of these fields manually after uploading the RTF into TeamConnect.

Examples of Basic Merge Fields

Data to be retrieved	Example merge field name	Sample output
Project name (system field)	`<<MatterName>>`	Company X vs. Company Y
Project number (system field)	`<<MatterNumber>>`	MATT-1234
Involved party selected in a custom field of type Involved	`<<OCAttorneyFirstName>>` `<<OCAttorneyLastName>>`	Kerry Goldman
Date from a custom field	`<<LossDate>>`	November 1, 2006
Dollar amount from a custom field of type Number	`<<EstimatedExpenses>>`	$4,000.00
Text that the user provides when generating the document	`<<SecretaryInitials>>`	jh
Current date	`<<Today>>`	November 21, 2006
Content from file (Such as LetterInfo.doc)	`<<InfoFile>>`	"This is a paragraph of information that is shared among several templates and

		is therefore managed as a separate file."
The current user's name	<<currentUserFirstName>> <<currentUserLastName>>	Jane Smith

1.1.21.2.2 Automatically Mapped System Fields

Using predefined merge field codes to automatically map system fields can save time and reduce errors in the configuration of your document template. When you open the RTF in the Data Mapping tool, TeamConnect does a lot of the mapping for you. See Locating the Data Mapping Tool for more information.

There are two ways to automatically map system fields:

- For certain fields, there is a predefined merge field code that will be recognized and automatically mapped. See Predefined Merge Field Codes.
- You can build the path of the system field in the merge field name. If you name the merge field with a valid path, it will be auto-mapped. See Naming Merge Fields for Auto-mapping System Fields.

You can also use predefined merge field codes to automatically map custom fields. See Automatically Mapped Custom Fields.

1.1.21.2.2.1 Predefined Merge Field Codes

The following table provides a list of merge field codes that automatically map to certain values, mostly having to do with assignees. These predefined codes make it easy to quickly map merge fields to assignees' contact information. This technique can also be used for custom fields of type Involved, as described in Automatically Mapped Custom Fields.

Note: *The technique behind the merge field codes for people is that the colon (:) is replacing the navigation.user.contact or .contact.*

Predefined Merge Field Codes

Object record type	System field	Predefined merge field code
Any record type (matters, tasks, involved parties, and so on)	Current date ***Note:*** *The date will be formatted using the format specified in System Settings. If you want to use a different format, do not automap the date.*	<<@date>>

	First name and last name of current user from his/her contact record	<<@user:firstName>> <<@user:name>>
	Name of user who created the record	<<@createdBy:firstName>> <<@createdBy:name>>
Task	Task assignee's first name and last name Note: These merge fields can be used inside of filter@ or loop@ merge fields that identify task records	<<@currentAssignee:firstName>> <<@currentAssignee:name>>
Any project (such as Matters)	Main assignee's name information from contact record	<<@mainAssignee:prefix>> <<@mainAssignee:firstName>> <<@mainAssignee:name>> <<@mainAssignee:suffix>> <<@mainAssignee:title>>
	Main assignee's contact ID	<<@mainAssignee:numberString>>
	Main assignee's default address from contact record	<<@mainAssignee:street>> <<@mainAssignee:city>> <<@mainAssignee:state>> <<@mainAssignee:postalCode>>
	Main assignee's default email address	<<@mainAssignee:email>>
	Main assignee's default phone	<<@mainAssignee:phone>>
	Main assignee's default fax	<<@mainAssignee:fax>>
	Main assignee's default Internet address	<<@mainAssignee:inetAddress>>
	Main assignee's default rate	<<@mainAssignee:defaultRateValue>>
	Main assignee's Social Security number or Tax ID	<<@mainAssignee:ss>>

1.1.21.2.2.2 Naming Merge Fields for Auto-mapping System Fields

You can cause system fields to be mapped automatically by following a naming convention in your merge field names:

```
@attributepath
```

where attributepath is the object attribute path that points from the starting object to the system field.

Object attributes are listed in Object Model: Read This First. To identify a system field that is related to the main object, create a path of attribute names that leads to the needed system field, delimited by periods (.).

Keep the following points in mind:

- If the system field is within the main object on which the template is based (such as the project number), then the path only contains one attribute with no periods.
- The final attribute cannot end with an arrow (->) in the attribute name as shown in Object Model: Read This First (or in related reference tables it points to) or the Object Navigator. For example, to select a contact, you must continue the path from contact-> to identify the name or some other attribute of the contact.
- You cannot create a path through an attribute that ends with List->, because these are related and sub-objects. For these, you must use loop@ or filter@ merge fields.
- System field names (known as object attributes) are case sensitive. Make sure to match the exact name and capitalization of the field name.

The following are examples of merge fields in a document template that is based on a project (such as Matter) that would be mapped automatically.

Examples of System Field Automatic Mapping

System field	Required merge field name	Sample output
Project name	`<<@name>>`	Company X vs. Company Y
Project number	`<<@numberString>>`	MATT-1234
Parent project name	`<<@parent.name>>`	Sorrelson Dispute
Opened On date of project	`<<@openedOn>>`	November 21, 2006
Name of contact selected in contact-centric field of project	`<<@contact.firstName>>` `<<@contact.name>>`	Joy Ferguson
Main assignee first name, last name	`<<@mainAssignee.contact.firstName>>` `<<@mainAssignee.contact.name>>`	Michael Ferguson
Main assignee's default phone number	`<<@mainAssignee.contact.defaultPhone.phoneString>>`	323-900-1700

Involved party's default role (when used inside of a filter@ merge field)	`<<@defaultCategory.name>>`	OC Attorney
Project's current phase	`<<@currentPhaseType.name>>`	Intake

1.1.21.2.3 Automatically Mapped Custom Fields

You can use predefined merge field codes to automatically map custom fields. This can save you time and reduce errors in the configuration of your document template. When you open the RTF in the Data Mapping tool, TeamConnect does a lot of the mapping for you. See Locating the Data Mapping Tool for more information.

Note that when a field is automatically mapped, the formatting is set to default options. For example, an automatically mapped date value will be formatted according to the default date format set in System Settings.

To automatically map a merge field to a custom field

1. Create a merge field with the following merge field name:

   `@Detail[`*`categoryTreePosition`*`]/`*`customFieldName`*

   where categoryTreePosition is the tree position of the category to which the custom field belongs, and customFieldName is the name (not the label) of the custom field.

   Examples:

   ```
   @Detail[CLMS_DEPA]/LiJudgeSettlemtDate
   @Detail[PRTP_0013]/PartyDate1
   ```

2. If the custom field is of type Involved, and you want to retrieve an attribute of the Involved's contact record, you can append a colon (:) and the name of the attribute, as described in the Predefined Merge Field Codes table.

   For example, if InvolvedOCAttorney is a custom field of type Involved where an involved OC attorney is identified:

   ```
   @Detail[MATT_LITI]/InvolvedOCAttorney:name
   @Detail[MATT_LITI]/InvolvedOCAttorney:firstName
   @Detail[MATT_LITI]/InvolvedOCAttorney:phone
   @Detail[MATT_LITI]/InvolvedOCAttorney:street
   @Detail[MATT_LITI]/InvolvedOCAttorney:city
   @Detail[MATT_LITI]/InvolvedOCAttorney:state
   @Detail[MATT_LITI]/InvolvedOCAttorney:postalCode
   ```

3. If the custom field is of type Custom Object, and you want to retrieve an attribute of the Custom Object record such as the name or number, you can append a period (.) and the name of the attribute.

   For example, if Litigation is a custom field of type Custom Object where a Litigation is identified:

```
@Detail[MATT_LITI]/Litigation.name
@Detail[MATT_LITI]/Litigation.numberString
```

4. If the custom field belongs to a related object of the main record for which the document template is created, nest the merge field within filter@ or loop@ merge fields. Otherwise, it does not need to be nested.

If you followed these steps correctly, the custom field is mapped automatically when you open the RTF in the Data Mapping tool in TeamConnect. See Locating the Data Mapping Tool for more information.

Tip: *If you prefer not to map the custom field automatically, try mapping it manually in the Data Mapping tool, as described in Custom Field Mapping.*

1.1.21.2.4 loop@ Merge Fields for Selecting Related or Sub-objects

If you need to include fields from sub-objects or related objects in the document template, and you do not want the user generating the document to select them from a list, then use loop@ in your merge field codes in the RTF file.

Note: *For users of Document Generator, loop@ corresponds to the Document Generator tag tc:loop for sub-objects and tc:search for related objects.*

The Data Mapping tool provides you with additional options for filtering the available sub-objects or related objects. For example, you may want to get the business address, not just any address listed for the contact. See Locating the Data Mapping Tool for more information.

In some cases, even with criteria that you specified, multiple sub-objects or related objects may be found. For example, a contact may have more than one business address. In this case, data from all of them will be listed in the generated document.

For details about how to perform mapping, see loop@ Merge Field Mapping.

Related and Sub-objects Available Through filter@ or loop@

Main object for which document template is defined	Related objects/Sub-objects	Object attribute in the Object Navigator
Any object (Contact, Matter, and so on)	History	historyList->
Project (such as Matters)	Assignees	assigneeList->
	Project relations (where related project is on left side of relationship)	leftRelationList->
	Project relations (where related project is on right side of	rightRelationList->

	relationship)	
	Involved	involvedList->
	Child or embedded projects	childList->
	Tasks	taskList->
	Appointments	appointmentList->
	Accounts (budgets)	accountList->
	Expenses	expenseList->
Contact	Addresses	addressList->
	Phone numbers	phoneList->
	Email addresses	emailList->
	Internet addresses	inetAddressList->
	Fax numbers	faxList->
	Territories	territoryList->
	Rates	rateList->
	Skills	skillList->
	Contact relations (where related contact is on the left side of relationship)	leftRelationList->
	Contact relations (where related contact is on the right side of relationship)	rightRelationList->
Invoice	Line items	lineItemList->

1.1.21.2.4.1 When NOT to Use loop@

Do not use loop@ under the following circumstances:

- If you want the user to manually select which sub-object's or related object's information to include in the generated document, you should use filter@. For details, see filter@ Merge Fields for User-selected Related or Sub-objects.

  If you want the user to manually select a sub-object or related object only when more than one is found, you should also use filter@.

- If you want to retrieve the address (or other sub-object) of the contact in a contact-centric project, you cannot use loop@. Instead, it is recommended to retrieve the default address of that contact, using a basic merge field that is mapped to an object field.

- If you want to get a sub-object that is set as "default," such as the default category, default address, or default phone number, you can do this with a basic merge field or an auto-mapped merge field, instead of a loop@ merge field. For examples, see Automatically Mapped System Fields.

- If you want to get the main assignee's name, you can do this with a basic merge field or with an automatically mapped field. See Automatically Mapped System Fields.

1.1.21.2.4.2 Inserting loop@ Merge Fields

To set up loop@ merge fields for sub-objects or related objects

1. In the RTF, insert a merge field with a name that starts with loop@ followed by the text of your choice.
2. Insert (or nest) basic merge fields for the data that you need to retrieve from the sub-object or related object, formatted the way you want them to appear in the generated document.
3. Complete the loop with a merge field named simply loop@.

For example:

```
The local office is located at the following address:
<<loop@partyAddress>><<street>>
<<city>>, <<state>> <<zip>>
<<loop@>>
```

This would generate the following:

```
The local office is located at the following address:
124 North Woodranch Road
Glen Oaks, CA 94551
```

For details about how to map these merge fields to TeamConnect data, see loop@ Merge Field Mapping.

1.1.21.2.5 filter@ Merge Fields for User-selected Related or Sub-objects

Using filter@ merge fields causes the document generator to pause at runtime and prompt the user to make a selection before the generator continues.

If you want the user to select related objects or sub-object(s) from which data must be retrieved, use filter@ merge fields. The Related and Sub-objects Available Through @filter or @loop table provides a list of related objects and sub-objects that the user can select for including data in the generated document.

When you use filter@ to define merge fields, you specify which related object records or sub-objects the user can select while generating the document. Data is then pulled into the document from the user-selected related records or sub-objects.

You can also nest a filter within another filter. For example, if you need the user to select a child project, and then select an assignee, you would nest one set of filter@ merge fields within another.

Note: *filter@ corresponds to the Document Generator tag tc:filter.*

1.1.21.2.5.1 When NOT to Use filter@

You should not use filter@ in the following circumstances:

- If you want to retrieve sub-object or related object data automatically, and you do not want the user to select which sub-object or related object to retrieve, use loop@. See loop@ Merge Fields for Automatically Selecting Related or Sub-objects.
- If you want to get information from the parent record of a project, use an automatically mapped merge field. See Automatically Mapped System Fields.
- If you want to get information from a record that is not related to the record for which a document is being generated, this is currently not possible with EasyDocs. Instead, you must create a Document Generator template manually using XML tags, as described in About Document Generator.

1.1.21.2.5.2 Inserting filter@ Merge Fields

To set up filter@ merge fields for related or sub-objects

1. In the RTF, insert a merge field with a name that starts with filter@ followed by the text of your choice.
2. Insert (or nest) basic merge fields for the data that you need to retrieve from the sub-object, formatted the way you want them to appear in the generated document.
3. Complete the filter with a merge field named simply filter@.

For example:

```
The following parties are participants in this matter:
<<filter@Participants>><<@contact.firstName>> <<@contact.name>>,
<<@defaultCategory.name>> <<@Detail[PRTP]/AgencyNumber>>
<<filter@>>
```

This would generate the following, if several related objects are retrieved:

```
The following parties are participants in this matter:
Reingold & Silverman, Outside Counsel Firm A-3433
John Silverman, Outside Counsel Attorney A-3434
Susan Yessler, Judge/Arbiter A-3420
```

For details about how to map these merge fields to TeamConnect data, see filter@ Merge Field Mapping.

1.1.21.2.6 if@ Merge Fields for Conditional Text

If you need the document generator template to include certain content only under a certain condition, use if@ merge fields.

The conditional content can include document content, such as text, and any other types of merge fields (basic, automatically mapped, loop@, and filter@ merge fields).

An if@ merge field can do any of the following:

- Check if a certain category has been added.
- Check if a system field has a value or not.
- Check if a custom field has a certain value.
- Check if a sub-object of a certain type has been added.

You define what the condition is that you need to check when you map the merge field to TeamConnect data.

Note: *if@ corresponds to the Document Generator tag tc:conditional.*

To set up if@ merge fields to identify conditional content

1. In the RTF, insert a merge field with a name that starts with if@ followed by the text of your choice.
2. Insert (or nest) basic merge fields for the content that you need to be generated only when the condition is met, formatted the way you want it to appear in the generated document.
3. Complete the filter with a merge field named simply if@.

For example:

```
<<if@LitigationCategoryAdded>> If the condition is true, you will see the content
of a related file.
Content of another file:
<<ImportedFileContent>>
<<if@>>
```

If true (that is, if the Litigation category exists), then the output in the generated document would be as follows:

```
If the condition is true, you will see the content of a related file.
Content of another file:
```

```
"This is a paragraph of information that is shared among several templates and is
therefore managed as a separate file."
```

For details about how to map the if@ merge field to define the condition, see if@ Merge Field Mapping.

1.1.21.3 Opening the Document Templates Folder

To upload RTF files to TeamConnect, and then map RTF merge fields to TeamConnect data, you must first locate the Documents Template folder.

The Document Templates folder can contain the following, as shown in the Document Templates Folder Example image:

- **RTF files**—The original files upon which the document templates are based.
- **DM files**—Files containing mapping information for an RTF file.

  Caution: *Do not modify or delete these files. They are automatically generated by the Data Mapping tool to store the mapping information, and should not be opened, modified,or touched.*

- **XML files**—Actual document templates used by the system when a user generates documents.

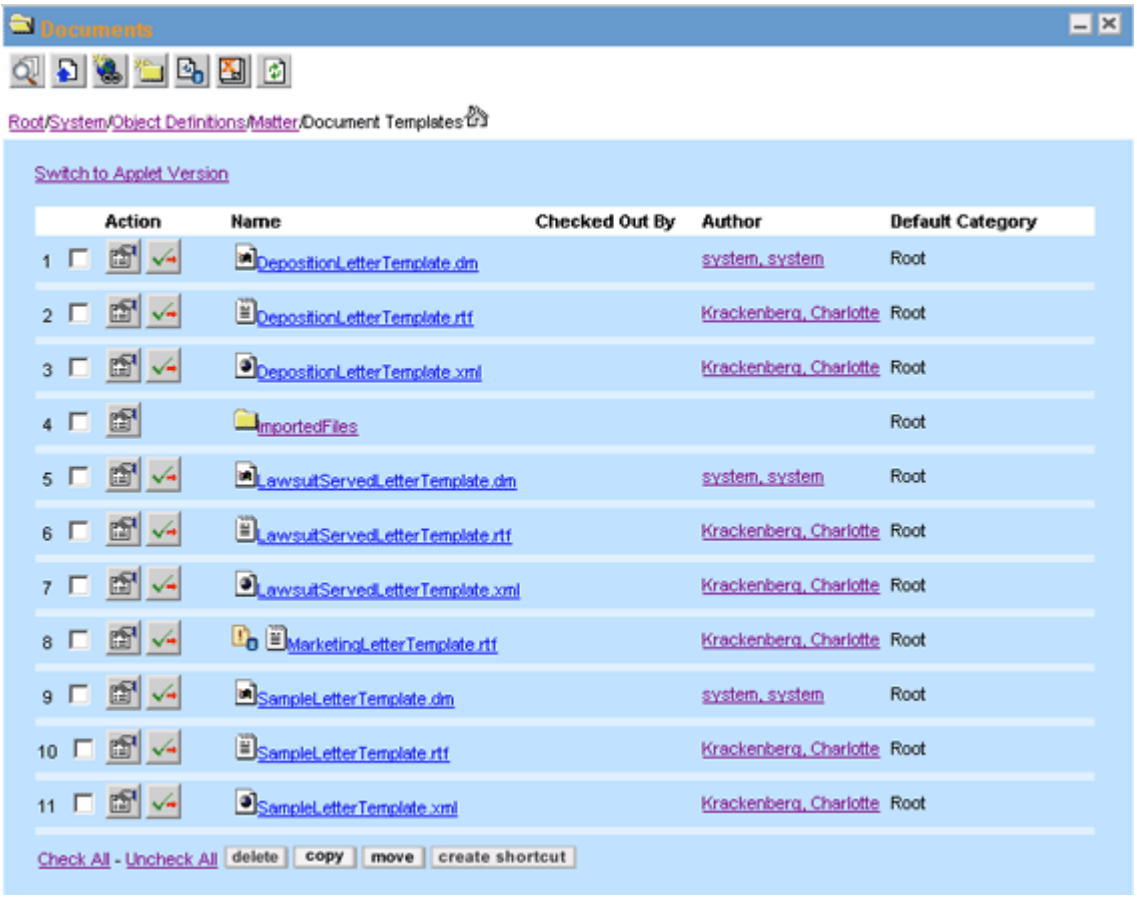
  Caution: *Do not modify or delete these files, unless they are created manually without the use of EasyDocs and have no corresponding RTF and DM files.*

- **Sub-folders**—Folders within the Document Templates folder for the object definition which can contain additional templates or files referenced by templates.

  Caution: *Once you choose a directory to hold your document template, do not rename, delete, or move that directory. Also, do not rename, delete, or move files within the directory. If you do, the associated documents no longer work correctly for end users at runtime.*

Locating the Data Mapping Tool

The Document Templates folder also contains the icon that opens the Data Mapping Tool.



Document Templates Folder Example

To open the Document Templates folder

1. In TeamConnect, click the **Setup** link.

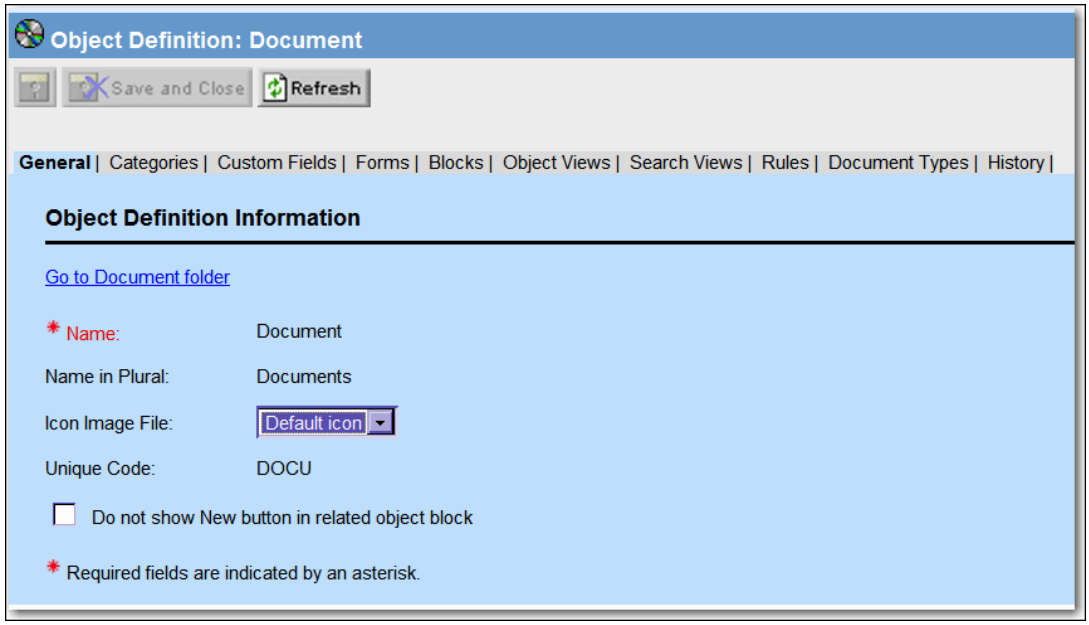
   The Designer opens.

2. In the **Go to** drop-down list, select **Object Definitions**.

   The Object Definition list view opens.

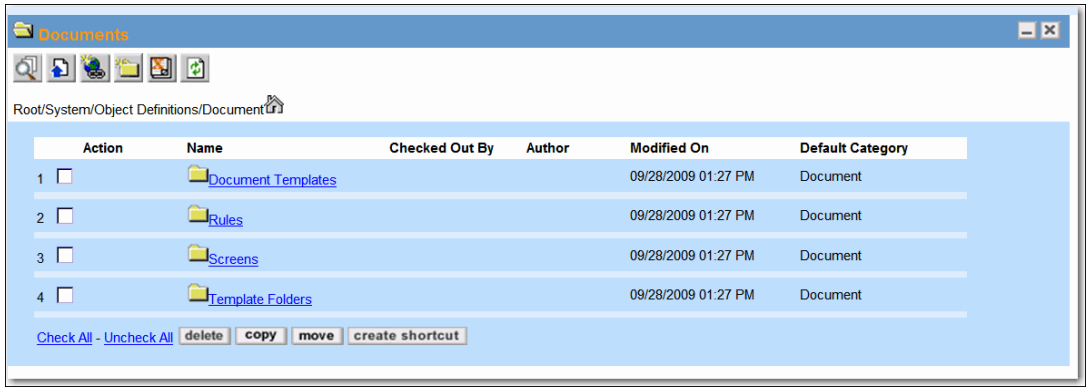
3. Click the Document object definition.

   The Document tab view opens to the **General** page.



Document Tab View

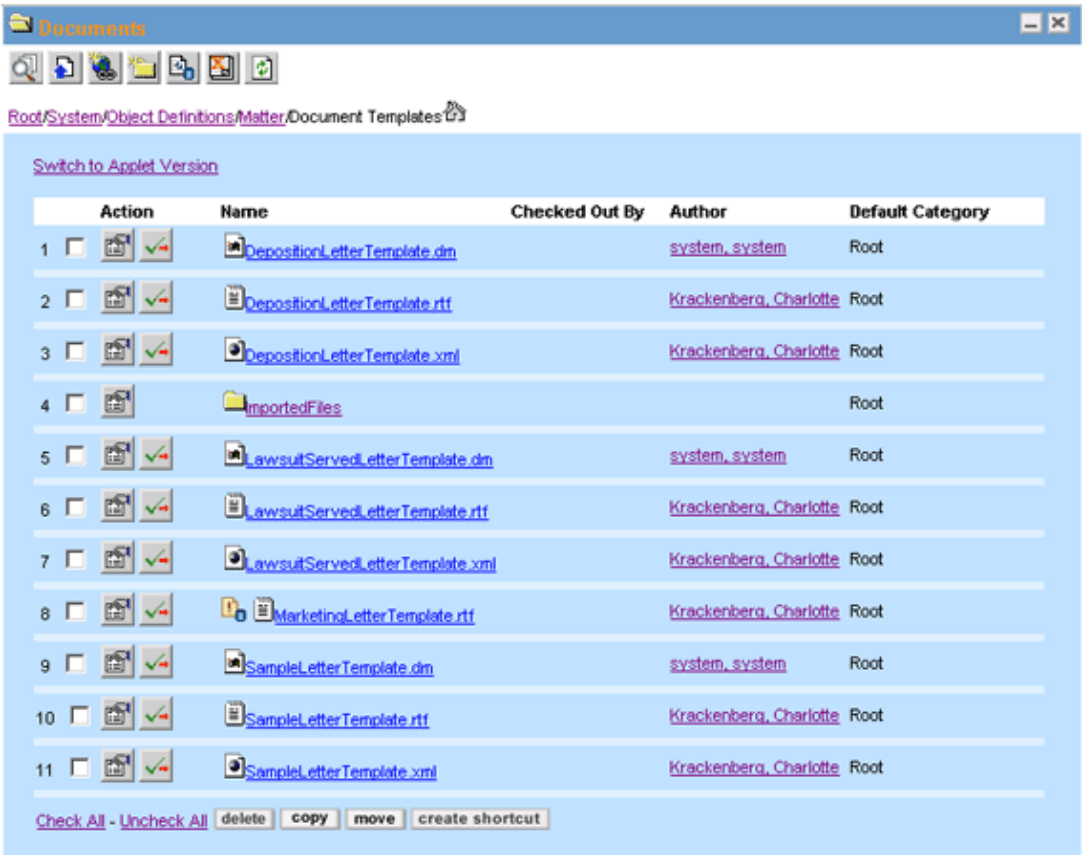
4. Click the **Go to Document folder** link.
5. The Document page opens and the **Document Templates** folder is listed.



Documents Page

6. To open the Document Templates folder, click the **Document Templates** link in the **Name** column.

   The **Document Templates** folder opens.



Document Templates Folder

Once you open the **Document Templates** folder, you can view the contents or select one of the following icons to perform a task.

Document Templates Folder Icons

Icon	Task
	Upload File
	Search Documents
	Create New Hyperlink

	Create Folder
	Launch Data Mapping Tool
	Launch Document Generator
	Refresh

1.1.21.4 Step 2: Upload RTF File to TeamConnect

Use this procedure to upload an RTF file to TeamConnect once it is ready to be mapped.

After uploading an RTF for the first time, you can modify it as needed while you are setting up mapping, for example, to modify your merge fields or document formatting. Use the **Check Out** and **Check In** buttons on the **Documents** screen to upload a modified version of the RTF. For details, see Modifying and Deleting Template Files.

To upload an RTF file into TeamConnect

1. Open the appropriate object definition and go to the object's folder.
2. Open the **Document Templates** folder. See Opening the Document Templates Folder for more information.

   Tip: *You can create subfolders inside of the **Document Templates** folder to organize your templates and then upload templates to those subfolders.*

3. Click **Upload File** on the toolbar.

   The **Upload New File** window opens.
4. Click **Browse** in the **Upload New File** window and find the RTF file that you want to upload.
5. Click **Upload File**.

   The **Upload New File** window closes.

   The RTF is listed on the screen, as shown in the Document Templates Folder Example image, with the **Not Data Mapped** icon next to it. This icon indicates that no mapping

has been saved for the RTF. After you perform data mapping, the icon is removed (even if the mapping is only partially completed).

After you upload the RTF file, you can map the merge fields in the RTF file to fields in TeamConnect.

1.1.21.5 Step 3: Map RTF Merge Fields to TeamConnect Data

Once you upload your RTF with merge fields into TeamConnect, it becomes available to be mapped in the Data Mapping tool. See Locating the Data Mapping Tool for more information.

Important: *In order to successfully map merge fields, you must be familiar with the Object Navigator. See Using Object Navigator.*

The techniques for mapping merge fields to TeamConnect data are described in the following procedures in this section:

- Opening an RTF in the Data Mapping Tool
- System Field Mapping
- Custom Field Mapping
- loop@ Merge Field Mapping
- filter@ Merge Field Mapping
- if@ Merge Field Mapping
- Mapping to Current Date
- Mapping to Current User
- Mapping to a File
- Defining a User Input Screen

1.1.21.5.1 Opening an RTF in the Data Mapping Tool

To open an RTF in the Data Mapping tool

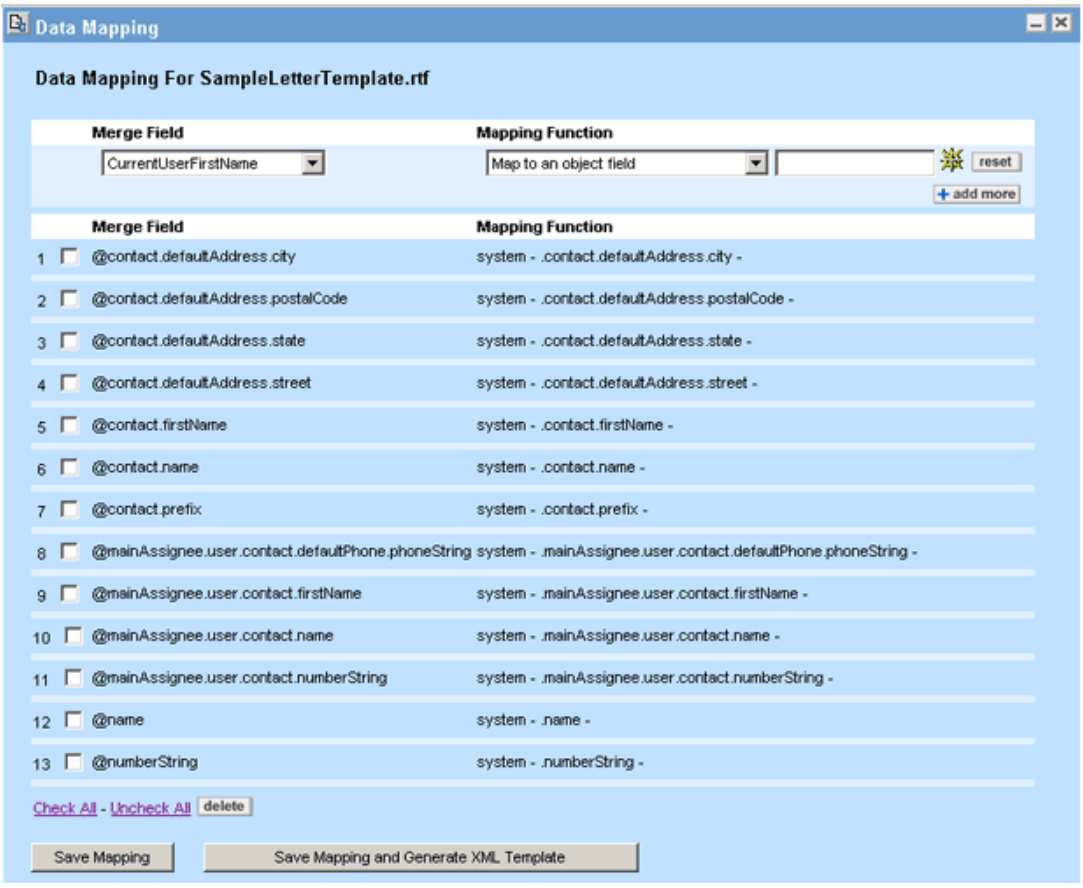
1. Navigate to the **Document Templates** folder or sub-folder where you uploaded the RTF. See Opening the Document Templates Folder for more information.
2. Click **Launch Data Mapping Tool** .

   The **Data Mapping** screen opens, with a list of all RTFs in the current **Document Templates** folder that contain unmapped merge fields.
3. If you do not see the button, make sure that you are in the HTML version of the **Documents** screen. Also, make sure that you have the right to the tool assigned to you, as listed in Required TeamConnect User Group Rights.

4. Click the name of the RTF that you want to map.

   The mapping screen of the Data Mapping tool opens, as shown in the following image.

   If you configured any merge fields to be automatically mapped to system fields, as described in Automatically Mapped System Fields, then the successfully mapped fields are already listed on the lower part of the **Data Mapping** screen.



Data Mapping Tool

1.1.21.5.2 System Field Mapping

There are two ways to map a merge field to a system field.

- If you correctly name the merge field as described in Automatically Mapped System Fields, then the mapping is done automatically when you open the RTF in the Data Mapping tool.
- If you created the merge field using your own name for it, you must perform the mapping in the Data Mapping tool.

See Locating the Data Mapping Tool for more information.

To map a basic merge field to a system field

1. Open the RTF in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.
2. Select the merge field that you want to map to a system field from the **Merge Field** drop-down list.

   The option **Map to an object field** is selected automatically as the **Mapping Function**.
3. Click the **Object Navigator** icon .

   The Object Navigator opens to the T-table of the object on which your template is based.

   For example, if the document template is for a custom object, such as Matter, then the Matter table appears.
4. Navigate to the system field value that you want to retrieve in the document using **traverse**.

   For commonly-used system field mapping, see the Navigation to Commonly-used System Fields table.

   Important: *Do not traverse through any attributes whose names end with List->. To get values from related objects or sub-objects, you must use filter@ or loop@ merge fields.*
5. Click ok when you reach the system field value. The object attribute cannot end with (->).
6. If necessary, select the formatting options provided in the Data Mapping tool for your selection.

   For example, if you are mapping the merge field to a field that stores a number, then the tool provides formatting options for numeric values.
7. Click **add more**.

   The merge field is mapped, and the mapping is listed in the lower section of the Data Mapping screen.
8. Click **Save Mapping** to save the mapping.

   Or, click **Save Mapping and Generate XML Template** to test the fields that you mapped. For details, see Step 5: Test Your Template.

The following table provides examples of how to navigate to certain commonly-used system field values for Document Generator templates if your starting object is a custom object (project), such as Matter.

Navigation to Commonly-used System Fields

System field	Navigation steps	Resulting path
Note: *These examples are for a document template that is based on a custom object.*		
Project Name	1 name (ok)	`.name`

Project Number	1 numberString (ok)	`.numberString`
Project's Default Category	1 defaultCategory-> (traverse) 2 name (ok)	`.defaultCategory.name`
Contact-centric contact's last name	1 contact-> (traverse) 2 name (ok)	`.contact.name`
Contact-centric contact's first name	1 contact-> (traverse) 2 firstName (ok)	`.contact.firstName`
Contact-centric contact's default address	Must have a separate merge field for each field in the address. Example navigation: 1 contact-> (traverse) 2 defaultAddress-> (traverse) 3 street (ok)	Examples: • `contact.defaultAddress.street` • `contact.defaultAddress.city` • `contact.defaultAddress.state` • `contact.defaultAddress.postalCode`
Main assignee's last name	1 mainAssignee-> (traverse) 2 user-> (traverse) 3 contact-> (traverse) 4 name (ok)	`.mainAssignee.user.contact.name`
Main assignee's first name	1 mainAssignee-> (traverse) 2 user-> (traverse) 3 contact-> (traverse) 4 firstName (ok)	`.mainAssignee.user.contact.firstName`
Main assignee's role	1 mainAssignee-> (traverse) 2 type-> (traverse) 3 name (ok)	`.mainAssignee.type.name`
Current Phase	1 currentPhaseType- > (traverse) 2 name (ok)	`.currentPhaseType.name`

A system field from the parent record of the project	1 parent-> (traverse) 2 Then, navigate as described for the needed parent system field.	Examples: • `.parent.numberString` • `.parent.mainAssignee.user.contact.name`
Name of an assignee with a certain role	Map a loop@ merge field to **assigneeList->**, as described in loop@ Merge Field Mapping. Or, map a filter@ merge field to **assigneeList->** if you want the user to select the assignee. See filter@ Merge Field Mapping.	
A system or custom field from a child record of the project (or from an embedded object record)	Map a filter@ merge field to **childList->**, as described in filter@ Merge Field Mapping.	

1.1.21.5.3 Custom Field Mapping

You can use this procedure to retrieve custom field values.

To map a merge field to a custom field

1. Open the RTF in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.
2. Select the merge field that you want to map to a custom field from the **Merge Field** drop-down list.

   The option **Map to an object field** is selected automatically as the **Mapping Function**.
3. Click the **Object Navigator** icon .

   The Object Navigator opens to the T-table of the object on which your template is based.

   For example, if the document template is for a custom object, such as Matter, then the Matter table appears.
4. Select **detailList->** and click **traverse**.

   The list of categories for the object appears.
5. Select the category to which the custom field belongs.
6. Click **traverse**.

   The list of custom fields for the selected category appears.
7. Select the name of the custom field.
8. If the custom field is of type Involved or Custom Object, click **traverse** and select the attribute that you want to use in the document (such as the project numberString).
9. Click **ok**.

The Object Navigator closes and the selected path appears in the Object Navigator field. One or more additional fields appear, which provide formatting options for the type of the custom field (such as date or number formatting options).

10. Select the appropriate formatting as you want it to appear in the generated document.
11. Click **add more**.

    The merge field is mapped, and the mapping is listed in the lower section of the **Data Mapping** screen.

1.1.21.5.4 loop@ Merge Field Mapping

Before mapping loop@ merge fields, make sure that you properly set up the merge fields that are nested within your two loop@ merge fields in the RTF, as described in loop@ Merge Fields for Automatically Selecting Related or Sub-objects.

There are two steps to mapping sub-objects or related objects to be automatically retrieved:

- Mapping loop@ Merge Field to Related Object or Sub-object
- Mapping Merge Fields Nested within loop@

The **Merge Field** drop-down list will not display the basic merge fields that are nested in your loop@ merge fields in until you map your loop@ merge field to a sub-object or related object.

1.1.21.5.4.1 Mapping loop@ Merge Field to Related Object or Sub-object

When you map the main loop@ merge field to a sub-object or related object, you are simply specifying the sub-object from which you want to automatically retrieve data.

To map a loop@ merge field to a sub-object or related object

1. Open the RTF in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.
2. Select the loop@ merge field that you want to map to a sub-object from the **Merge Field** drop-down list.

   The option **Map to an object field** is selected automatically as the **Mapping Function**.
3. Click the **Object Navigator** icon .

   The Object Navigator opens to the T-table of the object on which your template is based.

   For example, if the document template is for a custom object, such as Matter, then the **Matter** table appears.
4. Select the sub-object or related object attribute of the main object to which you want to map the loop@ merge field.

   For assistance identifying the object attribute that represents the sub-object or related object that you need, see the Related and Sub-objects Available Through filter@ or loop@ table.
5. Click **ok**.

The Object Navigator window closes and a second Object Navigator field appears, labeled **Qualifier**.

6. (Sub-objects only) If necessary, use the **Qualifier** field to identify a system field that you want to use to filter the list of sub-objects or related objects that are retrieved in the generated document.

   Typically, a list of sub-objects must be filtered by the sub-object type. For example, you may want to automatically list all assignees with the role Attorney.

   To select the type as the filter, use this navigation in the Object Navigator:

   a. **(Any)->** (traverse)

   b. **type->** (ok)

      The Object Navigator window closes and the path appears in the Qualifier field. A drop-down list of types (or roles) appears on the Data Mapping screen.

   c. Select the necessary type to use as a filter.

7. (Related objects only) If you are mapping loop@ to involved parties (involvedList->), you would typically use the **defaultCategory** as the qualifier. For child projects (childList->), the qualifier should be the specific custom object records you want displayed. For more assistance, see the steps in Mapping filter@ Merge Field to a Related or Sub-object.

8. Click **add more**.

   The merge field is mapped, and the mapping is listed in the lower section of the **Data Mapping** screen.

   The **Merge Field** drop-down list now includes the merge fields that are nested within the loop@ merge fields in the RTF. They are ready to be mapped to sub-object data.

1.1.21.5.4.2 Mapping Merge Fields Nested within loop@

After you map the main loop@ merge field, the nested merge fields become available in the Data Mapping tool.

To map nested merge fields to sub-object or related object data

1. Select one of the basic merge fields belonging to the loop@ from the **Merge Field** drop-down list.

   The option **Map to an object field** is selected automatically as the **Mapping Function**.

2. Click the **Object Navigator** icon .

   The Object Navigator opens, with the sub-object or related object already selected in the first table.

3. Navigate to the attribute of the sub-object or related object that you want to retrieve in the generated document.

The Sub-object Nested Merge Field Mapping Examples table lists examples of sub-object attributes that you may want to retrieve within loop@ and how to navigate to them.

4. Click **ok** to close the Object Navigator.

   The selected path appears in the Object Navigator field.

5. If necessary, select the formatting options provided in the Data Mapping tool for your selection.

   For example, if you are mapping the merge field to a field that stores a number, then the tool provides formatting options for numeric values.

6. Click **add more**.

   The merge field is mapped, and the mapping is listed in the lower section of the **Data Mapping** screen.

7. Click **Save Mapping** to save the mapping.

   The merge field is mapped, and the mapping is listed in the lower section of the **Data Mapping** screen.

Sub-object Nested Merge Field Mapping Examples

Sub-object	System field	Navigation steps	Resulting path
assigneeList->	Assignee last name	1 Select the assignee role (traverse) 2 user-> (traverse) 3 contact-> (traverse) 4 name (ok)	`.user.contact.name`
	Assignee first name	1 Select the assignee role (traverse) 2 user-> (traverse) 3 contact-> (traverse) 4 firstName (ok)	`.user.contact.firstName`
	Assignee role	1 Select the assignee role (traverse) 2 type-> (traverse) 3 name (ok)	`.type.name`
	Assignee's default phone number	1 Select the assignee role (traverse) 2 user-> (traverse)	`.user.contact.defaultPhone.phoneString`

		3 contact-> (traverse) 4 defaultPhone-> (traverse) 5 phoneString (ok)	
detailList->	Category name	1 name (ok)	`.name`
addressList->	Street	1 Select the address type (traverse) 2 street (ok)	`.street`
	City	1 Select the address type (traverse) 2 city (ok)	`.city`
	State	1 Select the address type (traverse) 2 state (ok)	`.state`
	Zip	1 Select the address type (traverse) 2 postalCode (ok)	`.postalCode`
phoneList->	Phone number	1 Select the phone type (traverse) 2 phoneString (ok)	`.phoneString`
faxList->	Fax number	1 Select the fax type (traverse) 2 faxString (ok)	`.faxString`

1.1.21.5.5 filter@ Merge Field Mapping

Every filter@ merge field will cause the letter generator to pause at runtime and prompt the user to select one or more choices from a list. The mapping determines what values that list will contain.

Before mapping filter@ merge fields, make sure that you properly set up the merge fields that are nested within your two filter@ merge fields in the RTF, as described in filter@ Merge Fields for User-selected Related or Sub-objects.

There are two steps to mapping related or sub-objects to be selected by the user:

- Mapping filter@ Merge Field to a Related or Sub-object

- Mapping Merge Fields Nested within filter@

The **Merge Field** drop-down list will not display the basic merge fields that are nested in your filter@ merge fields in until you map your filter@ merge field to a related object or sub-object.

1.1.21.5.5.1 Mapping filter@ Merge Field to a Related or Sub-object

When you map the main filter@ merge field to a related object or sub-object, you are simply specifying the related object (or sub-object) that you want the user to be able to select while generating a document with this template.

To map a filter@ merge field to a related or sub-object

1. Open the RTF in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.
2. Select the filter@ merge field that you want to map to a related or sub-object from the **Merge Field** drop-down list.

   The option **Map to an object field** is selected automatically as the **Mapping Function**.
3. Click the **Object Navigator** icon .

   The Object Navigator opens to the T-table of the object on which your template is based.

   For example, if the document template is for a custom object, such as Matter, then the Matter table appears.
4. Select the related object or sub-object of the main object to which you want to map the filter@ merge field.

   For assistance identifying the object attribute that represents the related object or sub-object you need, see the Related and Sub-objects Available Through filter@ and loop@ table.
5. Click **OK**.

   The Object Navigator window closes and a second Object Navigator field appears, labeled **Qualifier**. This field is used to identify a condition that you want to use to limit the list of related objects or sub-objects that the user can select, if needed.

   Do not specify an attribute that traverses to yet another object. You may specify simple attributes (number, text, date, boolean) or you may specify a list. If you specify a simple attribute, go to step 7.
6. For all lists, select **(Any)->** (traverse)

   Your next selection depends on what kind of list you chose. In most cases, you would select **defaultCategory->** (OK). For assigneeList, you would instead select **type->** (OK). For childList, you would select **application->** (OK), then select the child or embedded record type that you want the user to see at runtime.

   For example, if you are retrieving involved parties through involvedList->, you may want the list to include only involved parties with a certain role.

   To select the involved role as the **Qualifier**, use this navigation:

   a. **(Any)->** (traverse)

b. **defaultCategory->** (OK)

Object Navigator closes and the path is displayed in the Qualifier field. A drop-down list of roles is displayed on the Data Mapping screen.

c. Select the necessary role to use as a filter for which parties should appear.

7. Type the **Label** that you want the user to see when he or she is prompted to select the related or sub-object(s) when generating the document.
8. Select **Allow multiple selection** if you want the user to be able to select multiple items from the list when generating the document.
9. Select **Skip filter screen if only one record is found** if you want the end user to not have to make a selection when only one related or sub-object record is found. If this option is not selected, then the end user would see a drop-down list containing only one option and would have to click next.
10. Click **add more**.

The merge field is mapped, and the mapping is listed in the lower section of the **Data Mapping** screen.

The **Merge Field** drop-down list now includes the merge fields that are nested within the filter@ merge fields in the RTF. They are ready to be mapped to related object or sub-object data.

Once you generate the template, a user who generates a document with the template will be prompted to select a record before generating the document.

1.1.21.5.5.2 Mapping Merge Fields Nested within filter@

After you map the main filter@ merge field, the nested merge fields become available in the Data Mapping tool.

To map nested merge fields to related or sub-object data

1. Select one of the basic merge fields belonging to the filter@ from the **Merge Field** drop-down list.

The option **Map to an object field** is selected automatically as the **Mapping Function**.

2. Click the **Object Navigator** icon .

The Object Navigator opens, with the sub-object or related object already selected in the first table.

3. Navigate to the attribute of the related object or sub-object that you want to retrieve in the generated document.

The Sub-object Nested Merge Field Mapping Examples table lists examples of sub-object attributes that you may want to retrieve within filter@ and how to navigate to them.

The Related Object Nested Merge Field Mapping Examples table lists examples of related object attributes that you may want to retrieve within filter@ and how to navigate to them.

4. Click **ok** to close the Object Navigator.

   The selected path appears in the Object Navigator field.

5. If necessary, select the formatting options provided in the Data Mapping tool for your selection.

   For example, if you are mapping the merge field to a field that stores a number, then the tool provides formatting options for numeric values.

6. Click **add more**.

   The merge field is mapped, and the mapping is listed in the lower section of the **Data Mapping** screen.

7. Click **Save Mapping**.

Related Object Nested Merge Field Mapping Examples

Related object	Data to retrieve	Navigation steps	Resulting path
Any related object	Custom field value	1 (Any)-> (traverse) 2 detailList-> (traverse) 3 Select the category of the custom field (traverse) 4 Select the name of the custom field (ok)	Example: `.detailList(LAWY).detailDateValueList(MyDate2).detailValue`
Involved parties	Contact last name	1 (Any)-> (traverse) 2 contact-> (traverse) 3 name (ok)	`.contact.name`
	Contact first name	1 (Any)-> (traverse) 2 contact-> (traverse) 3 firstName (ok)	`.contact.firstName`
	Involved party's default role	1 (Any)-> (traverse) 2 defaultCategory-> (traverse) 3 name (ok)	`.defaultCategory.name`
Histories	Description (history text)	1 text (ok)	`.text`

Child projects (such as Participants) or embedded projects	Child project name	1 (Any)-> (traverse) 2 name (ok)	`.name`
	Child project number	1 (Any)-> (traverse) 2 numberString (ok)	`.numberString`
	Contact-centric contact's last name	1 (Any)-> (traverse) 2 contact-> (traverse) 3 name (ok)	`.contact.name`
	Contact-centric contact's first name	1 (Any)-> (traverse) 2 contact-> (traverse) 3 firstName (ok)	`.contact.firstName`
Appointments	Begins On date	1 (Any)-> (traverse) 2 startOn (ok)	`.startOn`
Tasks	Due on date	1 (Any)-> (traverse) 2 dueOn (ok)	`.dueOn`

1.1.21.5.6 if@ Merge Field Mapping

The steps for mapping an if@ merge field depend on what condition you want to check:

To use if@ to check if a category has been added to the record

Note: *You can check only categories of the main record for which a document is being generated.*

1. Open the RTF in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.
2. Select the if@ merge field that you want to map from the **Merge Field** drop-down list.

   The option **Map to an object field** is selected automatically as the **Mapping Function**.
3. Click the **Object Navigator** icon .

   The Object Navigator opens to the T-table of the object on which your template is based.

   For example, if the document template is for a custom object, such as Matter, then the Matter table appears.
4. Select **detailList->** and click **traverse**. Click **ok**.

The Object Navigator window closes and the path appears in the Object Navigator field. A drop-down list of categories appears.

5. Select the category that you want to check for existence in the record to determine whether the content within the if@ merge fields should be generated.
6. Select **Negate** if you want the content to be generated when the selected category is NOT present in the record. Otherwise, leave the check-box unchecked.
7. Click **add more**.
8. Click **Save Mapping**.

To use if@ to check of a system field is populated

1. Open the RTF in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.
2. Select the if@ merge field that you want to map from the **Merge Field** drop-down list.

   The option **Map to an object field** is selected automatically as the **Mapping Function**.
3. Click the **Object Navigator** icon .

   The Object Navigator opens to the T-table of the object on which your template is based.

   For example, if the document template is for a custom object, such as Matter, then the Matter table appears.
4. Navigate to the system field that you want to check for a value, and click **ok**.

   The Object Navigator window closes and the path appears in the Object Navigator field.
5. Select **Negate** if you want the content to be generated when the selected system field is NOT populated in the record. Otherwise, leave the check-box unchecked.
6. Click **add more**.
7. Click **Save Mapping**.

To use if@ to check if a custom field has a certain value

This method is most useful for custom fields of type List, Number, or Check Box.

Note: *You can check only custom fields belonging to the main record for which the document is being generated.*

1. Open the RTF in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.
2. Select the if@ merge field that you want to map from the **Merge Field** drop-down list.

   The option **Map to an object field** is selected automatically as the **Mapping Function**.
3. Click the **Object Navigator** icon .

The Object Navigator opens to the T-table of the object on which your template is based.

For example, if the document template is for a custom object, such as Matter, then the Matter table appears.

4. Select **detailList->** and click **traverse**.
5. Select the category of the custom field, and click **traverse**.
6. Select the custom field whose value you want to check and click **ok**.

   The Object Navigator window closes and the path appears in the Object Navigator field.
7. Select the value that you want to check for, depending on the field type. For example, if you want to check whether a certain lookup item is selected in the field, select that lookup item.
8. Select **Negate** if you want the content to be generated when the specified custom field does NOT have the specified value. Otherwise, leave the check-box unchecked.
9. Click **add more**.
10. Click **Save Mapping**.

To use if @ to check if a sub-object of a certain type has been added

You can use this mapping to check whether a sub-object of a certain type has been added, and thus specify whether content should be generated.

For example, if a certain type of assignee has been added, you may want to list the assignee's contact information in the generated document.

Note: *You can check only sub-objects of the main record for which a document is being generated.*

1. Open the RTF in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.
2. Select the if@ merge field that you want to map from the **Merge Field** drop-down list.

   The option **Map to an object field** is selected automatically as the **Mapping Function**.
3. Click the **Object Navigator** icon .

   The Object Navigator opens to the T-table of the object on which your template is based.

   For example, if the document template is for a custom object, such as Matter, then the Matter table appears.
4. Select the sub-object that you want to check (such as **assigneeList->**) and click **ok**.

   The Object Navigator window closes and the path appears in the Object Navigator field.
5. Select the type that you want to check for.
6. Select **Negate** if you want the content to be generated when the specified sub-object type is NOT added. Otherwise, leave the check-box unchecked.
7. Click **add more**.

8. Click **Save Mapping**.

1.1.21.5.7 Mapping to Current Date

A basic merge field can be set to the current date in the document when the user generates it. You can also specify the format of the date information.

Note: *This mapping function corresponds to the Document Generator tag tc:date.*

To map a merge field to the current date

1. Open the RTF in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.
2. Select the basic merge field that you want to map to the current date from the **Merge Field** drop-down list.
3. Select **Map to today's date** as the **Mapping Function**.

   Several additional fields for setting the date format appear.
4. Select the date format from the drop-down list.
5. Select **Show Time** if you want the time of the document generation to appear in the generated document, along with the date.
6. Select **Show Timezone** if you want the time zone of the current user to appear in the generated document, along with the date and time.
7. Click **add more**.

   The merge field is mapped, and the mapping is listed in the lower section of the **Data Mapping** screen.
8. Click **Save Mapping**.

The mapping is saved for the template.

1.1.21.5.8 Mapping to Current User

If you want to retrieve information about the current user who is generating the document, such as his or her name, you can do this by mapping a basic merge field to the current user.

Note: *This mapping function corresponds to the Document Generator tag tc:user.*

To map a merge field to the current user's first name

1. Open the RTF in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.
2. Select the basic merge field that you want to map to the current user from the **Merge Field** drop-down list.

3. Select **Map to current user** as the **Mapping Function**.
4. Click the **Object Navigator** icon .

   The Object Navigator opens to the YUser table.
5. Select **contact->** and click **traverse**.
6. Select **firstName** and click **ok**.

   The Object Navigator closes.
7. Click **add more** to add the mapping.

To map a merge field to the current user's last name

1. Open the RTF in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.
2. Select the basic merge field that you want to map to the current user from the **Merge Field** drop-down list.
3. Select **Map to current user** as the **Mapping Function**.
4. Click the **Object Navigator** icon .

   The Object Navigator opens to the YUser table.
5. Select **contact->** and click **traverse**.
6. Select name and click **ok**.

   The Object Navigator closes.
7. Click **add more** to add the mapping.

1.1.21.5.9 Mapping to a File

You can map a merge field to a file, so that the content of that file is imported into the generated document in place of the merge field. This is useful when the content is shared among multiple document templates.

If necessary, you can include merge fields that need to be mapped to TeamConnect data. In many cases, however, the external file contains only static text.

Mapping to a file requires two steps:

1. Converting RTF File to XML File—Any merge fields in the file must be mapped, and the file must be converted from RTF to XML.
2. Mapping to XML File

1.1.21.5.9.1 Converting RTF File to XML File

To convert an RTF file to an XML file in TeamConnect

1. Create a sub-folder in the **Document Templates** folder where the main RTF of your template is stored.

   Important: *You cannot map to a file unless it is stored in a sub-folder of the* ***Document Templates*** *folder of the template.*

2. In the sub-folder you created, upload the RTF file that you want to import into your template.
3. Click Launch Data Mapping Tool .

   The Data Mapping tool screen opens, where you can convert the RTF to XML.
4. Click the name of the RTF that you want to convert.
5. The mapping screen of the Data Mapping tool opens.

   If the file contains no merge fields, the text **(No Item Selected)** appears. If the file does contain merge fields that need to be mapped, they are listed in the **Merge Field** drop-down list.
6. If necessary, map all of the merge fields.
7. Click **Save Mapping and Generate XML Template**.

   A message appears that the generation is complete. The XML file is stored in the same folder as its RTF. You can now map a merge field in the main template to this file.

1.1.21.5.9.2 Mapping to XML File

If you already uploaded and converted the RTF to XML, you can map a merge field to it.

Note: *This mapping function corresponds to the Document Generator tag tc:file.*

To map a merge field to an external XML file

1. Open the main RTF in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.
2. Select the **Merge Field** that you want to map to a file.
3. Select **Map to a file** from the **Mapping Function** drop-down list.
4. Select the file to which you want to map the merge field.
5. Click **add more**.

   The merge field is mapped, and the mapping is listed in the lower section of the **Data Mapping** screen.
6. Click **Save Mapping**.

1.1.21.5.10 Defining a User Input Screen

When you map a merge field to user input, the Document Generator automatically creates a user input screen, so that the user generating the document can enter text to be included in place of the merge field.

Note: *This mapping function corresponds to the Document Generator tag tc:input.*

To define a user input screen

1. Open the main RTF in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.
2. Select the **Merge Field** that you want to map to a file.
3. Select **Define an input screen for user input** from the **Mapping Function** drop-down list.

   Additional fields appear for you to define the appearance of the input screen.
4. Enter the appropriate information to define the input screen:
   - **Screen Label**—Instructions to the user about what input to provide in the text box.
   - **Max Size**—Set the maximum number of characters for the input field, if you want the field to be only one line with no scrolling. This setting sets the length of the field, in characters, and also limits the text input to that number of characters.
   - **Show Scroll Bar**—Select this option if you want the user to enter text in a scrolling text box instead of a one-line text box.
   - **Rows**—When **Show Scroll Bar** is checked, this field sets the number of text rows of the text box that will be displayed. It does not limit the number of characters.
   - **Columns**—When **Show Scroll Bar** is checked, this field sets the width of the text box in the display, in characters. It does not limit the number of characters.
5. Click **add more**.

   The merge field is mapped, and the mapping is listed in the lower section of the **Data Mapping** screen.
6. Click **Save Mapping**.

Now, once you generate the template, a user who generates a document with the template will be prompted to enter text first before generating the document.

1.1.21.6 Step 4: Convert RTF File to a Document Generator Template

After you map some or all of the merge fields in your RTF file, you are ready to convert the RTF to a Document Generator template.

If you only mapped some of the merge fields, you can convert the RTF to a template in order to test your mapping. It is useful to do this several times in order to check your progress. Or you can wait until all merge fields are mapped to test your template.

To convert an RTF file to a Document Generator template

1. Open the RTF that you want to convert in the Data Mapping tool. For details, see Opening an RTF in the Data Mapping Tool.
2. Click **Save Mapping and Generate XML Template**.

   A message appears, indicating that the XML template has been generated.
3. Close the Data Mapping tool.

   The XML file has been placed in the same directory as the RTF file.

Now you are ready to test the template as described in Step 5: Test Your Template.

1.1.21.7 Step 5: Test Your Template

After mapping and generating a document template, you must ensure that it is retrieving data correctly by generating test documents.

Note: *Make sure that you have the appropriate rights, as listed in Required TeamConnect User Group Rights.*

To test a Document Generator template

1. Select one of the following options to browse to the Document Templates folder:
   - Using the **Documents** tab—Click the **Documents** tab. Click **More Actions**, and then click **Generate Documents**.
   - Using the **Documents** page for a record—Open the appropriate record and click the **Documents** link in the left pane. In the toolbar, Click the **Document Generator** icon.

     The **Document Generator** screen opens and displays a list of available templates.
2. Click the Select button for the template that you want to test.
3. Make any appropriate selections and enter text input as defined by your template.

   For example, your template may require the user to select related objects or sub-objects.
4. When the document generation is completed, click **Preview Document**.
5. Open or save the file.
6. Use the Successful Document Generation Checklist to verify that the document has been generated correctly.
7. If the document is correct, click **Save Document**.

For more information, see Generating Documents Using TeamConnect Data.

Successful Document Generation Checklist

Use the following checklist when testing your generated document:

- Make sure that you have adequate test data to use. For example, if the same contact records or data values are used repeatedly, or if values are missing, it may be difficult to distinguish whether data is being retrieved from the correct fields.
- Make sure that the document generation process prompts you for all user input, related object selections, and sub-object selections.
- View the generated document and make sure that there are no unpopulated merge fields.
- Make sure that all of the merge fields retrieve data from the correct sources (for example, user input and TeamConnect data). No merge field is skipped unless there is no data to retrieve from the specified source.
- If the template contains any if@ merge fields, make sure that the content inside the if@ merge fields is being generated only when the condition is met, and not when the condition is NOT met.
- If the template contains any filter@ merge fields, make sure that they are retrieving data from all related records selected by the user.
- If the template contains any loop@ merge fields, make sure that they are retrieving data from all of the correct related or sub-objects.

For more troubleshooting assistance, see About Document Generator and Troubleshooting Document Templates.

1.1.21.8 Modifying and Deleting Template Files

This section contains the following sub-sections:

- Modifying an Existing RTF File
- Modifying Existing Data Mapping
- Deleting Template Files

1.1.21.8.1 Modifying an Existing RTF File

You may need to modify the RTF of a template after you upload it, for example, to adjust the formatting or the merge field names, or to add or remove merge fields. You can modify an RTF without losing your mapping.

Caution: *Do not delete the RTF from the* ***Document Templates*** *folder, or you will lose your mapping and will have to re-map all of the merge fields in the RTF. Deleting the RTF will also automatically delete the XML template.*

To modify an existing RTF File

1. Navigate to the **Document Templates** folder or sub-folder where the RTF is stored in the **Documents** area of TeamConnect.
2. Check out the RTF. (Do not delete it, or you will lose all of the mapping.)
3. Modify the RTF as needed. If you modify merge fields, keep in mind:

- If you change the name of a merge field, it will no longer be mapped in the Data Mapping tool.

  The exception is if you change a merge field from a basic merge field with a custom name to an automatically mapped merge field (for example, @contact.name).

- If you remove a merge field, the mapping will automatically be removed in the Data Mapping tool.

- If you add a merge field, you will need to map it (unless it is an automatically mapped field).

4. Check in the RTF in the **Document Templates** folder or sub-folder.
5. Open the RTF in the Data Mapping tool (see Opening an RTF in the Data Mapping Tool).
6. If any additional mapping is needed, perform the data mapping.
7. Click **Save Mapping and Generate XML Template**.

The Document Generator template is generated, with the new formatting and/or data mapping.

1.1.21.8.2 Modifying Existing Data Mapping

You can modify the mapping of an existing template that has been created using EasyDocs.

To modify the data mapping of an existing document template:

1. Open the RTF of the document template in the Data Mapping tool. See Opening an RTF in the Data Mapping Tool.

   If there are any unmapped merge fields in the RTF, they are listed in the **Merge Field** drop-down list.

2. Map any unmapped merge fields.
3. To modify the mapping of an already mapped merge field:

   a. Select the mapped merge field's check-box and click **delete**.

   b. Click **OK** to confirm deletion of the mapping.

      The merge field appears as an unmapped merge field in the **Merge Field** drop-down list.

   c. Map the merge field.

4. Click **Save Mapping and Generate XML Template** when you are finished modifying the mapping.

The template now contains the revised mapping and is ready for you to test it.

1.1.21.8.3 Deleting Template Files

As in any Documents folder in TeamConnect, you have the option to delete files in the **Document Templates** folder (if you have the rights assigned to you).

Keep the following in mind when you consider deleting template files:

- If you delete an RTF from a Document Templates folder or sub-folder, both the DM mapping file and the XML template file (if you generated one) are also deleted and no longer available to end users for generating documents.
- If you delete an XML template file, the RTF and any completed data mapping are retained.

  However, the document template will not be available for generating documents, because the RTF file alone does not provide a document template to end users.

  You can generate a new XML template for an RTF using the Data Mapping tool.
- Deleting the DM file itself is not recommended. If you delete this file:
  - You will lose all data mapping between the RTF merge fields and TeamConnect data.
  - The **Not Data Mapped** icon will appear next to the RTF file, indicating that there is no mapping saved.
  - The XML template will not be deleted, and users can continue to generate documents using the template.
  - If you need to make any changes to the template, you will have to re-map the merge fields using the Data Mapping tool. See Locating the Data Mapping Tool for more information.

1.1.22 Troubleshooting Document Templates

The following table lists common issues you may encounter when creating document templates and how to resolve these issues:

Troubleshooting Document Templates

Issue	Resolution
Error when viewing XML file in Internet Explorer: Only one top level element is allowed in an XML document.	Make sure that the `tc:document` tag at the top is not closed and that at the very end of the document it includes the closing tc:document tag.
Error when viewing XML file in Internet Explorer: A string literal was expected, but no opening quote character was found.	Search and replace the following string with double quotes in your XML file. Replace \'94 with "

Error when viewing XML file in Internet Explorer: Reference to undeclared namespace prefix: 'tc'. Error processing resource.	You have not provided the `tc:document` tags and `tc:header` tags that are necessary for the template.
The web browser displays an error saying that the opening tag does not match the close tag. For example, `tc:content` does not match the opening tag `tc:data`. I've looked at my code and everything seems to be correct.	If you created your original document in Microsoft Word, it might be because the first tag on the very first line of your document was stored in the Title. 1. Open Microsoft Word and select File > Properties. 2. Check the Title field found in the Summary tab. If the Title field includes a tag, remove it. 3. Convert the RTF file to XML and try again. 4. If the title is not the problem, verify again that you have closed all of your tags in the correct sequence of nesting.
When generating the document I get an error message similar to: [*Error in Contact*]	When converting your document to XML, there may have been some issues that caused extra white space in the file. Review your XML file and make sure the Document Generator tags do not have any spaces or carriage returns inside them. Example #1: `<tc:data select="Contact">` Example #2: `<tc:data select="Contact">` To resolve this issue find <tc and check for any spaces or carriage returns inside the tag. Remove the spaces that were added while converting to XML so that it is on one line.
When generating the document I get an error message similar to: [*Error in DetailField: CONT_EMPL/ HireDate*] [*Error in numberString*]	• Make sure that the tree position is correct. The tree position must include the entire path (for example, Parent_Child). • Make sure that the field name is correct. • For system fields, make sure the first letter of the attribute name is uppercase.
When generating the document I get an error message similar to: No data provided for letter content.	There is probably a typographical error in your tc:document tag. Make sure the entire tag is typed correctly. For example, make sure that the xmlns attribute value is correct. This message also appears when there are errors in other tags. Make sure that all of your tags are typed correctly.

When I opened the generated document, tags were replaced by empty data and there were no errors, although there is data that should have been included. What is causing the problem?	• Make sure that the field corresponding to the tag has a value in the record. When a field has null value, the Document Generator will replace the tag with nothing. • Make sure that you (or the user generating the document) have rights to the objects, categories, and other data that you are retrieving in the template. Any data to which you do not have rights will not appear in the generated document. • Make sure that the tag and its attributes do not contain any extra characters (for example, {}). Example: `<tc:detail select="CLAM/}{ClaimEstimate}{" />` • Read the Issues in this table regarding errors when generating documents. If the setting is turned off to include errors in the generated document, then it may appear that data is not being retrieved when there is an error in a tag. • Your RTF file may be corrupted. To test it, create a new template only containing document generator tags (no text) and attempt to generate the new letter. If this template works, your old RTF file is corrupted and it should be replaced.
I uploaded the template, but when I try to test it by clicking on the Document Generator button, the template is not there.	• The template will not display unless it is well-formed. Make sure to open your template in a web browser before uploading it. • Make sure you are in an object record where that template is available when trying to test your template. • Make sure that you uploaded the template to the correct folder in the Documents area: o The template must be in the Document Templates folder for the correct Object Definition. o The template must not be in a subfolder of the Document Templates folder. Only supporting XML files for a template may be in a subfolder.
When I test my document template using Document Generator, the data from custom fields is not being inserted into the document.	• Make sure that you are testing using records where the data has been populated. • Make sure that you have included both the category tree position and the custom field name in the attribute value of the tag.

	• Make sure the category tree position represents the entire path to the category (for example, Parent_Child_Grandchild).
How do you find the tree position for custom and system lookup tables? For example tree position for address.	Each table found in the **Lookup Tables** screen has its own unique code. • For custom lookup tables, the unique code is defined specifically for your implementation of TeamConnect. It is located next to the name of the custom lookup table. • For system lookup tables, the unique code is defined by Mitratech.
When I open the generated document, the RTF source code appears instead of the actual document content.	This occurs when there is a line return or space after the opening `tc:content` tag. Make sure to include the RTF source code immediately after the opening `tc:content` tag.
When I test my document template using Document Generator, I get a format error similar to: [*Error in format: MMMMM-dd-YY*]	There is an error in the format that you have provided for a date field. For example, in this error, the Y's for the year are capitalized. Remember that the `dateFormat` tag attribute for date fields is case-sensitive.
When I test my document template using Document Generator, there is a huge chunk of code directly in the generated document where I used a tag to retrieve data.	You may have a tc:data tag that is trying to retrieve a data item of the data type "object." An object is not a valid type of data to retrieve in a document template. You may need to continue to navigate, for example, to the name attribute of the object. In Object Model: Read This First or in the reference tables it points to, locate the object attribute and find its data type in the Data Type column. If it is of type "object," then you cannot retrieve it using a tc:data tag. Refer to the object table to which that object attribute links to find the data that you need to retrieve in the document template.
When I test my document template using Document Generator, I get an error similar to one of the following: [*Error in numberString*] [*Error in parent*] [*Error in name*]	Notice that all of these errors specify an object attribute. The Document Generator is not recognizing them because the first letter is not capitalized. Make sure that every time you specify an object attribute in a tag, the first letter is capitalized.

A document is not generated correctly, and other means of troubleshooting fail.	If the debug level is activated, error information is written to the debug log. You may check this file for details on the problem.

The document template is generated successfully, but the letter generator hangs after selecting a document template.	At least one XML template is not well formed.
	1. Check for any un-escaped reserved characters (e.g. `&') in the XML template. If an un-escaped reserved character is found, please make the appropriate change in the XML template and test whether this resolves the issue.
	2. Check whether there are any begin/end tags missing from the XML template. For a migrated letter template, the missing begin/end tags are usually "filter@" or "loop@". Insert the missing begin/end tag mapping into the DM file and then use either Data Mapping Tool or EasyDocs Template Generator to re-generate the XML template.
	Here is the syntax and example for end-tag mapping:
	`<MergeField name="filter@"><operation>filter</operation>`
	`<ownerPath><MERGEFIELD begin tag name>::</ownerPath>`
	`<dataPath></dataPath>`
	`<conditionValue> </conditionValue>`
	`<formatOption></formatOption>`
	`</MergeField>`
	`<MergeField name="filter@"><operation>filter</operation>`
	`<ownerPath>filter@Name0.2::</ownerPath>`
	`<dataPath></dataPath>`
	`<conditionValue> </conditionValue>`

The XML template is generated successfully and the XML template is well formed, but launching Document Generator results in java.lang.StringIndexOutOfBoundException.	Open the XML template and check whether element <BASE_TREEPOSITION> or <TREEPOSITION> exists. If either one of them exists in the XML, replace the corresponding <BASE_TREEPOSITION> or <TREEPOSITION> with the appropriate TeamConnect Object Definition unique code or category tree position in the XML template. If you replace the unique code / tree position in the DM file, please use either Data Mapping Tool or EasyDocs Template Generator to regenerate the XML template. If the XML template does not contain either <BASE_TREEPOSITION> or <TREEPOSITION>,

	please report this issue to TeamConnect support for further assistance.

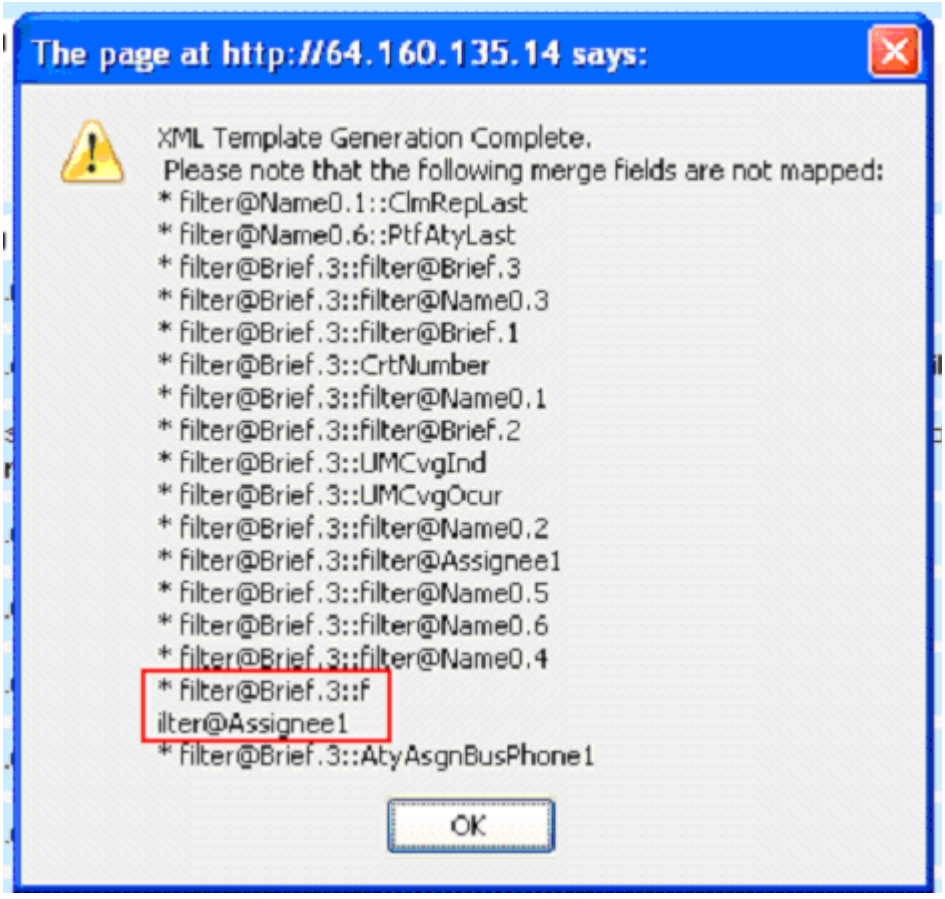
Additional Errors and Workarounds

If a line break is encountered in an XML file, in the middle of a token, it may cause an error message to be displayed. To fix the problems you must edit the XML file, removing the incorrect line breaks.

If the token is a merge field name, as shown below:

```
1523  \par }\pard \ltrpar\ql
      \li0\ri0\keep\keepn\widctlpar\tqr\tx9180\wrapdefault\aspalpha\aspnum\faauto\adjustright\rin0\lin0\itap0\pararsi
      {\field{\*\fldinst {\rtlch\fcs1 \af0 \ltrch\fcs0 \lang1024\langfe1024\noproof\insrsid5716630 MERGEFIELD f
1524  ilter@Assignee1 \\* MERGEFORMAT}}{\fldrslt {\rtlch\fcs1 \af0 \ltrch\fcs0 \lang1024\langfe1024\noproof\insrsid57
      \'abfilter@Assignee1\'bb}}}\sectd \pszl\pgnrestart\linex0\headery432\footery432\titlepg\sectdefaultcl\sectrsidl
```

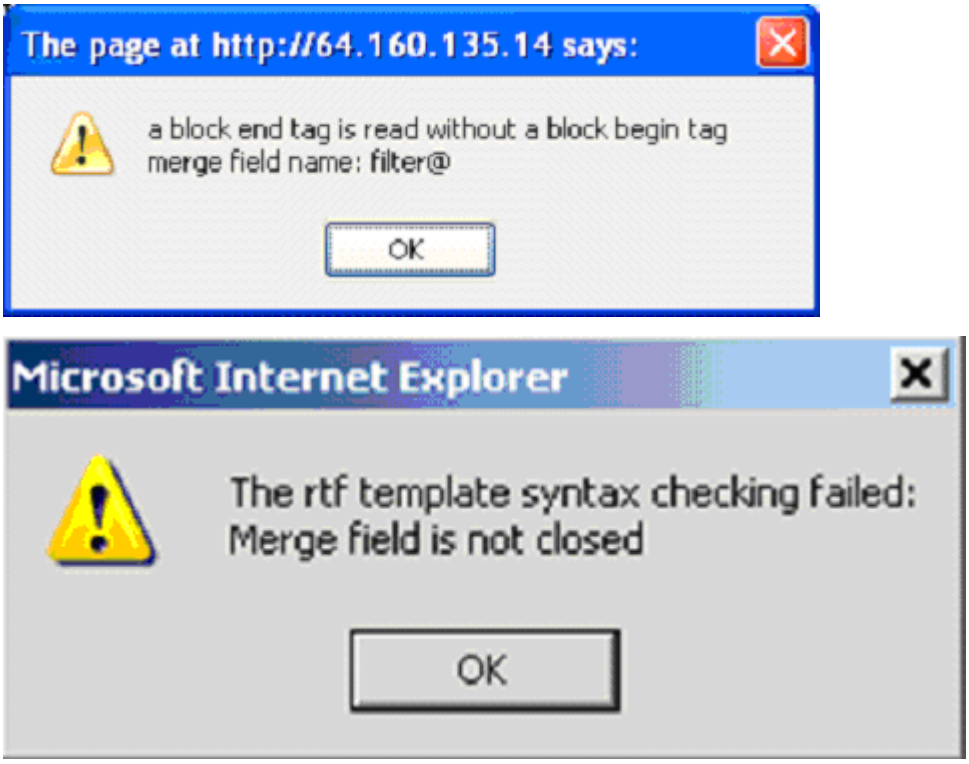
...then the resulting error message will look like this:



If the token containing the line break is a keyword instead, as shown below:

```
114  \af0\afs20\alang1025 \ltrch\fcs0 \fs24\lang1033\langfe1033\cgrid\langnp1033\langfenp1033 {\field
     \ltrch\fcs0 \insrsid10383591  MERGEFI¶
115  ELD  letterHead }}{\fldrslt {\rtlch\fcs1 \af0 \ltrch\fcs0 ¶
```

...then a different set of error messages will be displayed, as shown below:



1.1.23 Document Template Samples

Use the samples contained here as models for constructing tags in your document templates.

The following conventions are used in these samples:

Bold On the left of the sample, indicates what the code is doing or what data is being retrieved in that section of code.

Within the sample, indicates static text that would be in the content of the document.

`Code` Indicates tags that would be in the content of the document.

In some cases, comments are also written in code.

1.1.23.1 Nesting to Navigate

The following is an example of how to nest tags within system fields of data type `Object' to retrieve the necessary information from the main object. In this example, the main object is claim, which is a custom object.

As you will see, to retrieve data from a field of type `Object' you must specify the data that you want to retrieve by nesting the appropriate tags.

Contact-Centric

```
<!-- The contact-centric field is used to navigate to the
contact record. It is not used to retrieve specific data,
```

Field

such as the insured contact's first and last name. Therefore, we nested additional tags to retrieve the first and last name of the insured. -->

Contact: <tc:data select="Contact"><tc:data select="FirstName" /> <tc:data select="Name" /></tc:data>

Main Assignee

<!-- By nesting, we are able to navigate to the contact card of the main assignee to get the main assignee's name and phone number. This example also shows how to use tc:loop to get the business phone number as opposed to the default phone number. -->

Main Assignee: <tc:data select="MainAssignee">

First Name: <tc:data select="User"><tc:data select="Contact"><tc:data select="FirstName" />

Last Name: <tc:data select="Name" />

Business Phone Numbers: <tc:loop select="PhoneList" qualifier="Type_TreePosition=PHON_BUS1"><tc:data select="PhoneString" /></tc:loop>

</tc:data></tc:data></tc:data>

Created By

Created By: <tc:data select="CreatedBy"><tc:data select="Contact/FirstName" /> <tc:data select="Contact/Name" /></tc:data>

Current Phase Type

<!-- By itself, the object attribute CurrentPhaseType would not retrieve the name of the current phase. In order to do that, we needed to first navigate to WObjdPhaseType object to retrieve the actual name of the phase. The same is true for DefaultCategory. -->

Current Phase Type: <tc:data select="CurrentPhaseType"><tc:data select="Name" /></tc:data>

Default Category

Default Category: <tc:data select="DefaultCategory"><tc:data select="Name" /></tc:data>

Parent Policy

<!-- To retrieve information about the parent policy, we used the object attribute Parent to navigate to the parent policy and then nested the appropriate tags to retrieve specific data about the parent policy. -->

Parent Information: <tc:data select="Parent">

Name: <tc:data select="Name" />

Number: <tc:data select="NumberString" />

Main Assignee Info: <tc:data select="MainAssignee"><tc:data select="User/Contact/FirstName" /><tc:data select="User/Contact/Name" /></tc:data>

Assignees: <tc:loop select="AssigneeList"><tc:data select="User/Contact/FirstName" /> <tc:data select="User/Contact/Name" /> <tc:data select="Type/Name" /></tc:loop>

</tc:data>

1.1.23.2 Retrieving Data through Contact-Centric Fields

The following sample is going to retrieve data from a contact-centric system field found in a custom object definition.

The first tc:data tag navigates to the Contact object using the **contact** attribute of TProject. From there, the nested tags retrieve the following types of data about the contact.

- System fields
- Custom fields
- Sub-objects
- Related objects

Also, after navigating back to the main object (Claim) by closing the tc:data tag that navigated to the contact, the template retrieves data from a related object, Involved, using a filter defined in the header.

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:document version="1.0" mime-type="application/
msword" name="Contact-Centric Custom Fields"
xmlns:tc="http://www.w3.org/1999/XSL/Transform">
```

Creating Filter Screen for Related Objects

```
<!-- The header section identifies the filter screen
that will be displayed to the end-user. The filter
screen will display the Involved parties associated to
the Claim. The related object records in the filter
screen will be displayed in a drop-down list. -->
<tc:header>
<tc:filter name="SelectInvolvedParties"
searchCondition="Involved" label="Please select the
Involved Party" displayString="Contact/
FirstName;Contact/Name;DefaultCategory/Name"
displayFormat="*  * - *" />
</tc:header>
<tc:content><tc:data select="Contact">
```

First Name: `<tc:data select="FirstName" />`

Last Name: `<tc:data select="Name" />`

Retrieving Data from Custom Fields

```
<!-- The following tags retrieve data from the custom
fields within the Contact record. These custom fields
are defined for the category Employee. -->
```

Hire Date: `<tc:detail select="CONT_EMPL/HireDate" />`

Salary: `<tc:detail select="CONT_EMPL/Salary" />`

Employee Type: `<tc:detail select="CONT_EMPL/
EmployeeType" />`

Comments: `<tc:detail select="CONT_EMPL/Comments" />`

Citizen: `<tc:detail select="CONT_EMPL/Citizen"
true="Yes" false="No" />`

Book: `<tc:detail select="CONT_EMPL/Book"><tc:data
select="Name" /></tc:detail>`

Retrieving All Instances of the Sub-object Address

```
<!-- The following tags retrieve all of the addresses added to the contact record. In this case, the tc:loop tag is used because we want to retrieve all of the addresses added to the contact record, not just a specific type of address (which would require the qualifier attribute) or one that the user selected (which would require tc:filter to be used instead of tc:loop). -->
```

Address Information `<tc:loop select="AddressList">`

Street: `<tc:data select="Street" />`

Country: `<tc:data select="CountryItem/Name" />`

`</tc:loop>`

Automatically Retrieving All Instances of the Related Object History

```
<!-- The following tags retrieve all of the history records created for the contact record. In this case, the tc:search tag is used because we want to retrieve all history records that were created for the contact record, not just a specific history record (which would require the qualifier attribute or tc:conditional) or one that the user selected (which would require tc:filter to be used instead of tc:search). -->
```

History Information `<tc:search link="History">`

Archived On: `<tc:data select="ArchivedOn" />`

Default Category: `<tc:data select="DefaultCategory/Name" />`

Notes: `<tc:data select="Text" />`

`</tc:search></tc:data>`

Retrieving the User-Selected Related Object Involved

```
<!-- The following tags retrieve data pertaining to the involved party that the user selected while generating the letter. In this case, the tc:filter tag is used because we want to retrieve the data pertaining to the involved party that the user selected. -->
```

Involved Information `<tc:filter select="SelectInvolvedParties"><tc:data select="Contact">`

First Name: `<tc:data select="FirstName" />`

Last Name: `<tc:data select="Name" /> </tc:data>`

Automatically Retrieving All Instances of the Sub-Object Category

```
<!-- The following tags retrieve the categories added to the involved party record. In this case, the tc:loop tag is used because we want to retrieve all of the categories added to the user selected record. -->
```

Role: `<tc:loop select="DetailList"><tc:data select="Category/Name" />`

`</tc:loop></tc:filter></tc:content></tc:document>`

1.1.23.3 Automatically Retrieving Data from Related Objects

The following sample shows the different methods that can be used to automatically retrieve data from object records that are related to the main object (Claim). These samples retrieve data from:

- System fields from the main object (Claim) and its related object (Involved Parties)
- Sub-objects of the main object (Claim) and its related object (Involved Parties)

Claim Name: `<tc:data select="Name" />`

Claim Number: `<tc:data select="NumberString" />`

Insured: `<tc:data select="Contact"><tc:data select="FirstName" /> <tc:data select="Name" /> </tc:data>`

Retrieving All Instances of a Sub-object

```
<!-- The following tags retrieve data from all of the individuals assigned to the claim record. In this case, the tc:loop tag is used because we want to retrieve all of the assignees added to the claim record, not just a specific type of assignee (which would require the qualifier attribute) or one that the user selected (which would require tc:filter to be used instead of tc:loop). -->
```

Assignee Information `<tc:loop select="AssigneeList"> <tc:data select="User"> <tc:data select="Contact">`

First and Last Name: `<tc:data select="FirstName" /> <tc:data select="Name" />`

Phone Number: `<tc:data select="DefaultPhone/PhoneString" />`

Job Title: `<tc:data select="Title" />`

```
</tc:data></tc:data></tc:loop>
```

Retrieving Specific Type of Related Object with tc:conditional

```
<!-- The following is an example of how tc:conditional is used as a qualifier to retrieve the involved parties who are claimants. This will retrieve all of the involved records that have the role `Claimant' added.-->
```

Claimant Information `<tc:search link="Involved"><tc:conditional test="Detail[CLIN_CLMT]"><tc:data select="Contact">`

First Name: `<tc:data select="FirstName" />`

Last Name: `<tc:data select="Name" />`

Social Security #: `<tc:data select="SsOrTaxNumberString" />`

Driver's License: `<tc:data select="DriverLicense" />`

Company Name: `<tc:data select="Company"><tc:data select="Name" />`

Company Address: `<tc:data select="DefaultAddress"><tc:data select="Street" /> <tc:data select="City" /> <tc:data select="State" /> <tc:data select="PostalCode" /></tc:data></tc:data>`

Business Phone: `<tc:loop select="PhoneList" qualifier="Type_TreePosition=PHON_BUS1"><tc:data select="PhoneString" /> </tc:loop>`

Business Email: `<tc:loop select="EmailList" qualifier="Type_TreePosition=MAIL_BUS1"><tc:data select="EmailString" /></tc:loop>`

```
</tc:data></tc:conditional></tc:search>
```

Retrieving Specific Type of Related Object Using qualifier

```
<!-- The following is an example of how the qualifier attribute is
used to retrieve involved records that have the default role
`Injured Passenger' selected. -->
```

Injured Information `<tc:search link="Involved" qualifier="DefaultCategory_TreePosition=CLIN_CLMT_INPA"><tc:data select="Contact">`

First Name: `<tc:data select="FirstName" />`

Last Name: `<tc:data select="Name" />`

Social Security #: `<tc:data select="SsOrTaxNumberString" />`

Driver's License: `<tc:data select="DriverLicense" />`

Company Name: `<tc:data select="Company"><tc:data select="Name" />`

Company Address: `<tc:data select="DefaultAddress"><tc:data select="Street" /> <tc:data select="City" /> <tc:data select="State" /> <tc:data select="PostalCode" /></tc:data></tc:data>`

Retrieving Specific Types of Sub-objects

```
<!-- By using the qualifier attribute of tc:loop, we will retrieve
the contact record's business phone number(s) and email
address(es). -->
```

Business Phone: `<tc:loop select="PhoneList" qualifier="Type_TreePosition=PHON_BUS1"><tc:data select="PhoneString" /> </tc:loop>`

Business Email: `<tc:loop select="EmailList" qualifier="Type_TreePosition=MAIL_BUS1"><tc:data select="EmailString" /></tc:loop>`

`</tc:data></tc:search>`

1.1.23.4 Retrieving Data through Custom Fields of Type Custom Object

The following sample is used to retrieve data from the parent record of the Claim identified in a custom field of type Custom Object (MasterClaims).

`<tc:detail select="CLAM/MasterClaims">`

`<tc:data select="Parent">`

Master Claim's Parent (Policy) Information

Number: `<tc:data select="NumberString" />`

Name: `<tc:data select="Name" />`

Categories List: `<tc:loop select="DetailList">`

`<tc:data select="Category/Name" />`

`</tc:loop>`

Default Category: `<tc:data select="DefaultCategory/Name" />`

Policy Effective Date: `<tc:detail select="PLCY/EffectiveDate" />`

Good Driver?: `<tc:detail select="PLCY/GoodDriver" />`

Policy Postal Code: `<tc:detail select="PLCY/PolicyZip" />`

Policy State: `<tc:detail select="PLCY/PolicyState" />`

Comments: `<tc:detail select="PLCY/Comments" />`
`</tc:data> </tc:detail>`

1.1.23.5 Retrieving Data through Custom Fields of Type Involved

The following sample is used to retrieve data from the contact record linked to the project through the Involved custom field.

Additional Driver Info `<tc:detail select="PLCY/AdditionalDriver">`
`<tc:data select="Contact">`
First Name: `<tc:data select="FirstName" />`
Last Name: `<tc:data select="Name" />`
Home Address Information `<tc:loop select="AddressList" qualifier="Type_TreePosition=ADDR_BUS1"><tc:data select="Street" /><tc:data select="City" /><tc:data select="State" /><tc:data select="PostalCode" /></tc:loop>`
Social Security #: `<tc:data select="SsOrTaxNumberString" />`
Driver's License: `<tc:data select="DriverLicense" />`
`</tc:data></tc:detail>`

1.1.23.6 Retrieving Data from Child Objects

There are two tags used to retrieve data from child objects:

- `tc:search`, used to automatically retrieve data from child object records.
- `tc:filter`, used to define a screen where the end user selects the child object record(s) to be included in the document.

Automatically

The following sample is used to retrieve data from all child claims of a parent policy. In addition, it retrieves the vehicles damaged in the accident of each claim. Vehicle is an embedded object of Claim. Notice that tc:search is nested within another tc:search tag.

Policy Information

Policy Number: `<tc:data select="NumberString" />`
Insured: `<tc:data select="Contact"><tc:data select="FirstName" /> <tc:data select="Name" /></tc:data>`

Retrieving Specific Type of Child Records with tc:conditional

```
<!-- The tc:conditional tag is used to retrieve child claim records of type `Auto'. -->
<tc:search link="Project" isForChild="yes" qualifier="Application_ProjObjectDefinition_UniqueCode=CLAM"> <tc:conditional test="Detail[CLAM_AUTO]">
```

Claim Information (Child of Policy)

Claim Name: `<tc:data select="Name" />`

Claim Number: `<tc:data select="NumberString" />`

Assignees: `<tc:loop select="AssigneeList" qualifier="IsActive=1"><tc:data select="User"> <tc:data select="Contact">`

Name, **Date Assigned**

`<tc:data select="FirstName" /> <tc:data select="Name" /></tc:data></tc:data>, <tc:data select="AssignedOn" />`

`</tc:loop>`

`<tc:data select="MainAssignee">`

Main Assignee: `<tc:data select="User"><tc:data select="Contact"><tc:data select="FirstName" /> <tc:data select="Name" /></tc:data></tc:data></tc:data>`

`<!-- Retrieving vehicles damaged in each claim -->`

`<tc:search link="Project" isForChild="yes" qualifier="Application_ProjObjectDefinition_UniqueCode=VEHI">`

Vehicle Information `<!-- Embedded Object of Claim -->`

`The following vehicles were involved in the this claim:`

VIN: `<tc:detail select="VEHI/VIN" />`

Make: `<tc:detail select="VEHI/MakeModel" />`

Year: `<tc:detail select="VEHI/Year" />`

Damage: `<tc:detail select="VEHI/Damage" />`

`</tc:search></tc:conditional></tc:search>`

User Selected

This sample utilizes the tc:filter tag to retrieve data from the user selected child record(s). This sample retrieves data from:

- System fields from policy record.
- User-selected assignee(s) of the policy record.
- User-selected claim record.
- User-selected assignee(s) of the selected claim record.
- User-selected involved contact(s) of the selected claim record.

Header

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:document version="1.0" mime-type="application/msword" name="Policy, Claim and Vehicle Information - Filter" xmlns:tc="http://www.w3.org/1999/XSL/Transform">
<tc:header>
<tc:filter name="SelectedPolicyAssignee" test="AssigneeList" label="Please select the assignee to contact in regards to the Policy"
```

```
displayString="User/Contact/FirstName;User/Contact/Name;Type/Name"
displayFormat="* * - *" />
<tc:filter name="Claims" searchCondition="Project"
qualifier="Application_ProjObjectDefinition_UniqueCode=CLAM"
label="Select the appropriate claim" displayString="Name;NumberString"
displayFormat="* - *" usedBy="2" />
<tc:filter name="SelectedClaimAssignee" test="AssigneeList"
label="Please select the assignee to contact in regards to the Claim"
displayString="User/Contact/FirstName;User/Contact/Name;Type/Name"
displayFormat="* * - *" multi="yes" objectFrom="Claims"/>
<tc:filter name="SelectedInvolvedParties" searchCondition="Involved"
label="Please select the parties involved in this Claim"
displayString="Contact/FirstName;Contact/Name;DefaultCategory/Name"
displayFormat="* * - *" multi="yes" objectFrom="Claims"/>
</tc:header>
```

Content

```
<tc:content>
```

Policy Number: `<tc:data select="NumberString" />`

Insured Information `<tc:data select="Contact">`

First Name: `<tc:data select="FirstName" />`

Last Name: `<tc:data select="Name" /> </tc:data>`

```
<!-- The following tags retrieve data from the Assignee that the user
selected. In this case, the tc:filter tag is used because we want to
retrieve data from the sub-object that the user selected, not just all
of the users assigned to the policy record (which would require the
tc:loop) or specific type of assignee (which would require the
qualifier attribute). -->
```

Assignee Information `<tc:filter select="SelectedPolicyAssignee"><tc:data select="User"><tc:data select="Contact">`

First Name: `<tc:data select="FirstName" />`

Last Name: `<tc:data select="Name" /></tc:data></tc:data>`

Role: `<tc:data select="Type/Name" /></tc:filter>`

```
<!-- The following tags retrieve data from the user selected claim
record(s), which is a child object of Policy. In addition, we nested
other tags to retrieve data pertaining to the related objects and sub-
objects of Claim. -->
<tc:filter select="Claims">
```

Claim Number: `<tc:data select="NumberString" />`

Assignees `<tc:filter select="SelectedClaimAssignee"><tc:data select="User"><tc:data select="Contact">`

First Name: `<tc:data select="FirstName" />`

Last Name: `<tc:data select="Name" /></tc:data></tc:data>`

Assigned On: `<tc:data select="AssignedOn" />`

Role: `<tc:data select="Type/Name" /></tc:filter>`

Main Assignee: `<tc:data select="MainAssignee"><tc:data select="User"><tc:data select="Contact"><tc:data select="FirstName" /> <tc:data select="Name" /></tc:data></tc:data></tc:data>`

```
<!-- The following tags retrieve data from the involved contact
selected by the user. -->
```

Involved Parties `<tc:filter select="SelectedInvolvedParties"><tc:data select="Contact">`

First Name: `<tc:data select="FirstName" />`

Last Name: `<tc:data select="Name" /></tc:data>`

Roles: `<tc:data select="DefaultCategory/Name" /></tc:filter></tc:content></tc:document>`

1.1.23.7 Using Qualifiers to Filter Sub-objects

This sample shows how you can use the qualifier attribute of tc:loop to refine the data that you want to retrieve. It also shows how you can include different types of system fields as a qualifier.

This template is retrieving all of the active assignees within the selected claim. For each active assignee, it is retrieving:

- The tasks that they perform where the rate is equal to $100.00.
- Their skill level as an adjuster.
- Their address within the city of Los Angeles.
- The contacts that have a vendor relationship with them.

Boolean (Check Box)

`<tc:loop select="AssigneeList" qualifier="IsActive=1">`

`<tc:data select="User"> <tc:data select="Contact">`

Name: `<tc:data select="FirstName" /> <tc:data select="Name" /></tc:data></tc:data>`

Role: `<tc:data select="Type/Name" />`

Date Assigned: `<tc:data select="AssignedOn" />`

`<tc:data select="User"> <tc:data select="Contact">`

Number

`<tc:loop select="RateList" qualifier="RateAmount=100">Rate Information`

Task Type `<tc:data select="TaskCategory/Name" />`

Rate `<tc:data select="RateAmount" />`

Effective From `<tc:data select="EffectiveFrom" />`

Effective To `<tc:data select="EffectiveTo" />`

`</tc:loop>`

List (Lookup Table)

`<tc:loop select="SkillList" qualifier="Type_TreePosition=SKIL_ADST">Skill Information`

Skill `<tc:data select="Type/Name" />`

Level of Expertise `<tc:data select="SkillLevel" />`

`</tc:loop>`

Text

```
<tc:loop select="AddressList" qualifier="City=Los Angeles">Address Information
Type <tc:data select="Type/Name" />
Street <tc:data select="Street" />
City <tc:data select="City" />
State <tc:data select="State" />
Postal Code <tc:data select="PostalCode" />
Country <tc:data select="CountryItem/Name" />
Current On <tc:data select="CurrentOn" />
</tc:loop>
```

Object (Another Record)

```
<tc:loop select="RightRelationList" qualifier="RightContact_DefaultCategory_TreePosition=CONT_VNDR">Contact Relations
<tc:data select="RightContact">
Name: <tc:data select="FirstName" /> <tc:data select="Name" /></tc:data>
Relationship: <tc:data select="Type/Name" />
<tc:data select="RightContact">
Business Address: <tc:loop select="AddressList" qualifier="Type_TreePosition=ADDR_BUS1">
Street <tc:data select="Street" />
City <tc:data select="City" />
State <tc:data select="State" />
Postal Code <tc:data select="PostalCode" />
</tc:loop>
</tc:data>
</tc:loop>
```

1.1.23.8 Using tc:conditional

There are many different ways to use `tc:conditional` to retrieve data, include static text, and/or import text from other files.

1.1.23.8.1 Comparing Custom Fields

The following are examples of how `tc:conditional` can be used to compare a custom field value with the value specified in the tag to include static text and retrieve data.

Date

```
<tc:conditional condition="greater" compare="Detail[CLAM]/DetailValue[AwardDate],2003-01-01">Date Awarded: <tc:detail select="CLAM/AwardDate" /></tc:conditional>
```

Number

```
<tc:conditional condition="less" compare="Detail[CLAM]/DetailValue[ClaimEstimate],10000">Claim Estimate: <tc:detail select="CLAM/ClaimEstimate" /></tc:conditional>
```

List

```
<tc:conditional compare="Detail[CLAM]/
DetailValue[Fatality],Yes">Unfortunately there was a fatality.</
tc:conditional>
```

Check Box

```
<tc:conditional compare="Detail[CLAM]/
DetailValue[PoliceInvolved],true">Police Involved: <tc:detail
select="CLAM/PoliceOfficer" /></tc:conditional>
```

Text

```
<tc:conditional compare="Detail[CLAM]/DetailValue[LossLocation],Near
Residence">Loss Location: <tc:detail select="CLAM/LossLocation" /></
tc:conditional>
```

1.1.23.8.2 Comparing Sub-objects

The following example will check to see if an assignee with the role `Supervisor' is added to the claim. If so, it will include the information nested within the `tc:conditional` tag. If not, nothing will be included.

```
<tc:conditional compare="System[Assignee],Adjuster">Please contact your claim
adjuster <tc:loop select="AssigneeList"
qualifier="Type_TreePosition=CLAM_ADST"><tc:data select="User"><tc:data
select="Contact"><tc:data select="FirstName" /> <tc:data select="Name" /> at
<tc:data select="DefaultPhone/PhoneString" /></tc:data></tc:data>.</tc:loop></
tc:conditional>
```

The following example will loop through all of the assignees within the claim record to see if any of the assignees have the skill 'Adjuster' in their contact card. If so, it will include the information nested within the `tc:conditional` tag. If not, nothing will be included.

```
<tc:loop select="AssigneeList" qualifier="Type_TreePosition=PHON_BUS1"><tc:data
select="User"><tc:data select="Contact"><tc:conditional
compare="System[Skill],Adjuster">Please contact <tc:data select="FirstName" />
<tc:data select="Name" /> at <tc:loop select="PhoneList"
qualifier="PHON_BUS1"><tc:data select="PhoneString" /></tc:loop> if you have any
questions regarding your claim.</tc:conditional></tc:data></tc:data></tc:loop>
```

1.1.23.8.3 Testing for Existing Data

The following samples show how `tc:conditional` is used to see if data exists for system fields or sub-objects. If data exists, the nested information is included. If not, nothing is included.

Testing System Fields

The following example will test to see if the claim is closed. If so, it will include the static text and the date that the claim was closed.

```
<tc:conditional test="ClosedOn">Your claim was closed on <tc:data select="ClosedOn" />
.</tc:conditional>
```

The following example will test to see if the claim is linked to a policy. If so, it will include the static text and the policy number.

```
<tc:conditional test="Parent">This claim is in regards to Policy Number <tc:data
select="NumberString" />.</tc:conditional>
```

Testing Sub-objects

The following example will test to see if any assignees are assigned to the claim. If so, it will include the static text and loop to find all the assignees with the role `Adjuster' and bring in the nested information.

```
<tc:conditional test="AssigneeList">Please contact your claim adjuster <tc:loop
select="AssigneeList" qualifier="Type_TreePosition=CLAM_ADST"><tc:data
select="User"><tc:data select="Contact"><tc:data select="FirstName" /> <tc:data
select="Name" /> at <tc:data select="DefaultPhone/PhoneString" /></tc:data></
tc:data></tc:loop>if you have any questions.</tc:conditional>
```

The following example will test to see if the claim includes the category `Auto'. If so, it will include the static text in the letter and retrieve data from the specified system and custom fields.

```
<tc:conditional test="Detail[CLAM_AUTO]">We have the following information about your
auto accident.
Date of Accident: <tc:detail select="CLAM_AUTO/AccidentDate" />
Loss Location: <tc:detail select="CLAM/LossLocation" />
Claimant: <tc:detail select="CLAM_AUTO/Claimant" />
Policy Number: <tc:data select="Parent"><tc:data select="NumberString" /></
tc:data></tc:conditional>
```

1.1.23.8.4 Including Information When Certain Conditions Are Met

The following sample shows how `tc:conditional` can be used to include the appropriate pronoun based on the gender of an individual. In this sample, the pronoun he/she or his/her is based on whether the claimant is male or female.

```
<tc:document version="1.0" mime-type="application/msword"
name="Conditional" xmlns:tc="http://www.w3.org/1999/XSL/Transform">
```

Header

```
<tc:header>
<tc:filter name="UserSelectedClaimant" searchCondition="Involved"
displayString="Contact/FirstName;Contact/Name;DefaultCategory/Name"
displayFormat="* * - *" label="Please select the Claimant"/>
</tc:header>
```

Content

```
<tc:content>
<tc:data select="Contact"><tc:data select="FirstName" /> <tc:data
select="Name" />
<tc:loop select="AddressList"><tc:data select="Street" />
<tc:data select="City" />, <tc:data select="State" />  <tc:data
select="PostalCode" /></tc:loop>
Dear <tc:data select="Prefix" /> <tc:data select="FirstName" />
<tc:data select="Name" /></tc:data>
```

We were notified of your recent auto accident with `<tc:filter select="UserSelectedClaimant"><tc:data select="Contact"><tc:data select="FirstName" /> <tc:data select="Name" />.`

`<tc:conditional compare="Detail[CONT]/DetailValue[Gender],Male">He</tc:conditional><tc:conditional compare="Detail[CONT]/DetailValue[Gender],Female">`**She**`</tc:conditional></tc:data></tc:filter>` **has reported that the accident occurred on** `<tc:detail select="CLAM_AUTO/AccidentDate" />.`

Based on `<tc:filter select="UserSelectedClaimant"><tc:data select="Contact"><tc:conditional compare="Detail[CONT]/DetailValue[Gender],Male">his</tc:conditional><tc:conditional compare="Detail[CONT]/DetailValue[Gender],Female">her</tc:conditional></tc:data></tc:filter>` **report the following people were involved in the auto accident.**

```
<tc:search link="Involved"><tc:data select="Contact"><tc:data
select="FirstName" /> <tc:data select="Name" /></tc:data> - <tc:data
select="DefaultCategory/Name" />
</tc:filter>
```

Sincerely,

```
<tc:loop select="AssigneeList"
qualifier="Type_TreePosition=CLAM_ADST"><tc:data select="User"><tc:data
select="Contact"><tc:data select="FirstName" /> <tc:data
select="Name" /></tc:data></tc:data></tc:loop>
</tc:document>
```

1.1.24 Using Tag Attributes

This appendix provides examples of XML tag attributes in TeamConnect that are commonly used to format fields and their labels on the screen. The attributes are grouped by the type of custom field each attribute affects. For details on the field and label tags, Block Tags.

Field Tags

The field tags that can be formatted with the help of attributes include:

Form	`<TCFIELD NAME="FieldName" attributes />`
Block	`<tc:field name="FieldName" category="TreePosition" attributes />`
	`<tc:wizardParameter name="ParameterName" attributes />`

To include an attribute in one of these tags, replace attributes with the attributes and their values. For example, to restrict the length of a field (MomMaidenName) to 20 characters, you would use the maxlength attribute:

```
<TCFIELD NAME="MomMaidenName" maxlength="20" />
```

Label Tags

Unlike fields that can have a variety of formatting attributes, field labels may have only a the formatting attributes listed in Field Label Tag Attributes. The label tags are:

Form
```
<TCLABEL NAME="FieldName" attributes />
```

Block
```
<tc:label name="FieldName" category="TreePosition" attributes />

<tc:wizardParameterLabel name="ParameterName" attributes />
```

To include an attribute in one of these tags, replace attributes with the attributes and their values. For example, if you do not want the settings defined for required fields to be applied to a field (MomMaidenName), you would use the notUseRequiredFieldStyle attribute, as in the following example:

```
<TCLABEL NAME="MomMaidenName" notUseRequiredFieldStyle="true" />
```

1.1.24.1 Field Label Tag Attributes

You can use the following attributes with custom field and parameter labels:

- notUseLabelStyle
- notUseRequiredFieldStyle
- irequired
- style

notUseLabelStyle

Set this attribute to "`true`" if you do not want the **Label Text** style defined in the system settings or user preferences for both Edit and Read-only mode to be applied to a custom field or parameter label.

You can then use a style sheet or the `style` attribute to define the font color and size of the label. If you do not use a style sheet or the style tag, the field label is set to the **General Text** settings defined in system settings or user preferences.

Form
```
<TCLABEL NAME="LastReviewDate" notUseLabelStyle="true"/>
```

Block
```
<tc:label name="LastReviewDate" category="EMPL" notUseLabelStyle="true" /
>

<tc:wizardParameterLabel name="LastReviewDate" notUseLabelStyle="true" />
```

notUseRequiredFieldStyle

Set this attribute to "`true`" if you do not want the **Required Field** style defined in system settings or user preferences to be applied to a label of a required custom field or parameter.

You can then use your style sheet or the `Style` tag to define the font color and size of the label. If you do not use a style sheet or the Style tag, the field label is set to the **General Text** settings defined in system settings or user preferences.

Form
```
<TCLABEL NAME="LastReviewDate" notUseRequiredFieldStyle="true" />
```

Block
```
<tc:label name="LastReviewDate" category="EMPL"
notUseRequiredFieldStyle="true" />

<tc:wizardParameterLabel name="LastReviewDate"
notUseRequiredFieldStyle="true" />
```

isFieldRequired

Set this attribute to "`true`" if you want to make a Label Only field required, for example, when it is associated with another required field. you can also use this attribute to make the label of a custom field required when this field is made required through a rule or its attribute is included in the unique ID pattern for custom object records.

Form
```
<TCLABEL NAME="LastReviewDate" isFieldRequired="true" />
```

Block
```
<tc:label name="LastReviewDate" category="EMPL" isFieldRequired="true" />

<tc:wizardParameterLabel name="LastReviewDate" isFieldRequired="true" />
```

For details on system settings, see Working with Admin Settings.

Note: *To learn more about the style sheets, refer to an HTML or XML reference guide.*

1.1.24.2 Tag Attributes for All Field Types

forceNotEditable

Set this attribute to "`true`" to display custom fields as Read-only text. For example:

Form
```
<TCFIELD NAME="LastReviewDate" forceNotEditable="true"/>
```

Block
```
<tc:field name="LastReviewDate" category="EMPL"
forceNotEditable="true" />

<tc:wizardParameter name="LastReviewDate" forceNotEditable="true" />
```

style

The `style` attribute enables you to apply specific properties such as color, font, text alignment to a field. The `style` attribute also applies to the field label tags (see Label Tags).

Points to remember:

- Properties defined in the `style` attribute override those defined in the system settings, user preferences, or by the `style` tag.

- You can apply all style properties available in cascading style sheets.
- When defining the `style` attribute you must adhere to the following requirements:
  - Include the attribute value within braces ({}).
  - Separate each style with a semicolon (;).
  - Separate each property and its value with a colon (:).
- The style applies to both Edit and Read-only modes.

For example:

Form
```
<TCFIELD NAME="LastReviewDate" style="{text-align:right;font-style:italic}" />
```

Block
```
<tc:field name="LastReviewDate" category="EMPL" style="{text-align:right;font-style:italic}" />

<tc:wizardParameter name="LastReviewDate" style="{text-align:right;font-style:italic} />
```

tabIndex

By default, the tabbing order is set to tab from left to right, top to bottom. You can, however, customize the tabbing order to meet the needs of your users by using the tabIndex attribute with each field on the screen. Specify the order for each field, beginning with 1.

Points to remember:

- If this attribute is used, it must be added to every field tag in your form or custom block.
- Fields must not have duplicate `tabIndex` values.
- Applying `tabIndex` to the following field types affects the way the user navigates to their corresponding buttons.
  - *Custom Object* and *Involved* field types, which have **Find** and **Open** buttons.
  - *Date* field type, which has a **Calendar** button.

  The user is unable to navigate to these buttons by using the TAB key. He or she is required to use the mouse.
- Applying `tabIndex` to a Date field that has the Time field displayed, affects the way the user navigates to the time field. The user is unable to navigate to the time field by using the TAB key. He or she is required to use the mouse.
- The **Detail Forms** system block or custom block that includes the `tabIndex` attribute must be added as a separate tab within an object view.

  This is because the system blocks do not have the tabIndex attribute. A screen that includes multiple blocks where they contain this attribute and others do not, prevents the user from using the TAB key through the entire screen.

- You can include multiple custom blocks to a tab within an object view, if the tab order is sequential and adheres to all of the requirements noted above.

For example:

Form
```
<TCFIELD NAME="LastReviewDate" tabIndex="1" />
```

Block
```
<tc:field name="applicationEntity.LastReviewDate" category="EMPL"
tabIndex="1" />

<tc:wizardParameter name="LastReviewDate" tabIndex="1"/>
```

1.1.24.3 Number Field Tag Attributes

format

Controls the format of the number field to either dollar, decimal or percent. The following values can be entered:

- DOLLAR—Sets the format of the number field to include the appropriate currency sign, a decimal point, and commas. For example, `42000` is formatted to `$42,000.00` after the user saves the record.

  Form
  ```
  <TCFIELD NAME="AnnualSalary" format="DOLLAR" />
  ```

  Block
  ```
  <tc:field name="AnnualSalary" category="EMPL" format="DOLLAR" />

  <tc:wizardParameter name="AnnualSalary" format="DOLLAR" />
  ```

  This attribute displays the symbol of the system default currency in all object records except contacts, where it displays the symbol of the currency selected for that contact.

  When used for custom fields of type Number in invoices that are entered in the original currencies different from the system default currency, this attribute will not reflect the currency symbol of the invoice.

- DECIMAL—Sets the format of the number field to include a decimal point and commas. The number of zeros after the decimal point depends on the values of two other attributes:
  - `minimumFractionDigits` is used in cases where you wish to show digits to the right of the decimal point even when the rightmost decimal place(s) contain zero.
  - `maximumFractionDigits`. is used in cases where you wish to store no more than a specified number of decimal places. If the user types more than the specified number of decimals, the result is rounded to the specified number of decimals.

  If either of these attributes is not specified, the attribute's default number of decimal places is 2.

  The value of `maximumFractionDigits` must not be specified as less than the value of `minimumFractionDigits`.

  Form
  ```
  <TCFIELD NAME="Salary" format="DECIMAL" />
  ```

Block
```
<tc:field category="MATT" name="Rate" format="DECIMAL"
minimumFractionDigits="3" maximumFractionDigits="4" />
```

For example, if the user types 42000 into field "Salary", above, the user input `42000` is formatted to `42,000.00` (default 2 decimal places) after the user saves the record.

An additional example: In the "Rate" field shown above, if the user types 37.1, it is formatted at save time to 37.100 (the minimum of three decimals applies in this case.) If the user types 37.10046, the field is formatted to 37.1005 (the maximum of four decimals applies in this case, and rounding is used.)

- NONE—Sets the format of the number field to use no formatting, such as a decimal point and commas. For example, if a number field is set to NONE, the user input `1501082006` is not formatted with commas and appears exactly as it is entered.

Form
```
<TCFIELD NAME="VoucherNumber" format="NONE" />
```

Block
```
<tc:field name="VoucherNumber" category="VNUM" format="NONE" />

<tc:wizardParameter name="VoucherNumber" format="NONE"/>
```

- PERCENT—Sets the format of the number field to round the decimal to the nearest integer and to include the percent sign. For example, if a number field is set to `PERCENT` the user input `45.75` is formatted to `46%` after the user saves the record.

Form
```
<TCFIELD NAME="RaisePercent" format="PERCENT" />
```

Block
```
<tc:field name="RaisePercent" category="EMPL" format="PERCENT" />

<tc:wizardParameter name="RaisePercent" format="PERCENT" />
```

size

Controls the width of number fields by specifying the number of numeric characters for the field. For example:

Form
```
<TCFIELD NAME="Salary" size="10" />
```

Block
```
<tc:field name="Salary" category="EMPL" size="10" />

<tc:wizardParameter name="Salary" size="10" />
```

maxlength

Limits the maximum number of characters a user can enter in number fields. Set the attribute value to a number to represent the maximum number of numeric characters allowed. For example:

Form
```
<TCFIELD NAME="Salary" size="10" maxlength="10" format="DOLLAR" />
```

Block
```
<tc:field name="Salary" category="EMPL" size="10" maxlength="10"
format="DOLLAR" />

<tc:wizardParameter name="Salary" size="10" maxlength="10"
format="DOLLAR" />
```

onFocus="javascript:blur();"

Prevents users from entering data into the number fields. This attribute is used with business rules and JavaScript. For example:

Form
```
<TCFIELD NAME="Percent" format="PERCENT" onFocus="javascript:blur();" />
```

Block
```
<tc:field name="Percent" category="EMPL" format="PERCENT"
onFocus="javascript:blur();" />

<tc:wizardParameter name="Percent" format="PERCENT"
onFocus="javascript:blur();" />
```

The same rules apply to including multiple attributes for a tag. For example, the custom field Salary can have the attributes `format`, `size`, and `maxlength`. See the example in maxlength.

1.1.24.4 Text Field Tag Attributes

size

Sets the width for text fields. Set the attribute value to a number to represent the width of the field in characters. For example:

Form
```
<TCFIELD NAME="LastEmployer" size="50" />
```

Block
```
<tc:field name="LastEmployer" category="EMPL" size="50" />

<tc:wizardParameter name="LastEmployer" size="50" />
```

maxlength

Sets the maximum number of characters for text fields. Set the attribute value to a number to represent the maximum number of alphanumeric characters allowed. For example:

Form
```
<TCFIELD NAME="Age" size="3" maxlength="3" />
```

Block
```
<tc:field name="Age" category="EMPL" size="3"
maxlength="3" />

<tc:wizardParameter name="Age" size="3" maxlength="3"/>
```

1.1.24.5 Memo Text Field Tag Attributes

rows

Sets the height of memo text fields by specifying the number of rows for the memo text field. For example:

Form
```
<TCFIELD NAME="Comments" rows="10" />
```

Block
```
<tc:field name="Comments" category="EMPL" rows="10" />

<tc:wizardParameter name="Comments" rows="10" />
```

cols

Sets the width of memo text fields by specifying the number of columns for the memo text field. Enter a number to represent the width. For example:

Form
```
<TCFIELD NAME="Comments" cols="125" />
```

Block
```
<tc:field name="Comments" category="EMPL" cols="125" />

<tc:wizardParameter name="Comments" cols="125" />
```

wrap

Sets how line breaks are managed in memo text fields. The following values can be entered:

- OFF—Restricts the browser from wrapping text, which forces the user to do it manually.
- SOFT or VIRTUAL—Eliminates all the line break symbols from the element's value after the form is submitted. `SOFT` is only supported by Netscape Navigator browsers. `VIRTUAL` is only supported by Microsoft Internet Explorer browsers.
- HARD or PHYSICAL—Maintains all the line break symbols in the element's value when the form is submitted. `HARD` is only supported by Netscape Navigator browsers. `PHYSICAL` is only supported by Microsoft Internet Explorer browsers.

For example:

Form
```
<TCFIELD NAME="Comments" wrap="OFF" />
```

Block
```
<tc:field name="Comments" category="EMPL" wrap="OFF" />

<tc:wizardParameter name="Comments" wrap="OFF" />
```

1.1.24.6 Date Field Tag Attributes

showTime

Displays the time field to the right of date fields. Set the attribute value to "true" to have the time field appear next to the date field. For example:

Form
```
<TCFIELD NAME="DateHired" showTime="true" />
```

Block
```
<tc:field name="DateHired" category="EMPL" showTime="true" />

<tc:wizardParameter name="DateHired" showTime="true" />
```

timeUnderDate

Displays the time field directly below the date fields. Set the attribute value "true" to have the time field appear below the date field. For example:

Form
```
<TCFIELD NAME="DateHired" showTime="true" timeUnderDate="true" />
```

Block
```
<tc:field name="DateHired" category="EMPL" showTime="true"
timeUnderDate="true" />

<tc:wizardParameter name="DateHired" showTime="true"
timeUnderDate="true" />
```

Important: *You must include the showTime attribute to use timeUnderDate.*

size

Controls the width of date fields by specifying the number of characters for the field. For example:

Form
```
<TCFIELD NAME="DateHired" size="8" />
```

Block
```
<tc:field name="DateHired" category="EMPL" size="8" />

<tc:wizardParameter name="DateHired" size="8" />
```

1.1.24.7 List Field Tag Attributes

type

Turns a list field, which by default is a drop-down list, into a radio button list. Enter "`radio`" to set the list field as a radio button list. For example:

Form
```
<TCFIELD NAME="PerformanceReview" type="radio" />
```

Block
```
<tc:field name="PerformanceReview" category="EMPL" type="radio" />
```

Note: *You can define wizard parameters of type List as drop-down, radio, or check box lists (see Defining Parameters).*

size

Controls the height of list fields by specifying the number of rows in the list to be displayed. For example:

Form
```
<TCFIELD NAME="Gender" size="20" />
```

Block
```
<tc:field name="Gender" category="EMPL" size="20" />

<tc:wizardParameter name="Gender" size="20" />
```

allowNullValue

Includes a null value as the first item in a drop-down list. Enter "true" to include a null value in the list. That way, no item is selected by default.

Important: *The allowNullValue attribute must be set to use nullValueDisplayString.*

nullValueDisplayString

Specifies the first item displayed in a drop-down or radio button list. Enter a message that should appear as the first item in the list, such as **Choose One**. For example:

Form
```
<TCFIELD NAME="Gender" allowNullValue="true"
nullValueDisplayString="Please Select One" />
```

Block
```
<tc:field name="Gender" category="EMPL" allowNullValue="true"
nullValueDisplayString="Please Select One" />

<tc:wizardParameter name="Gender" allowNullValue="true"
nullValueDisplayString="Please Select One" />
```

forceSelect and forceSearch

If **forceSelect=true**, the field will display a drop-down list when used as a search view filter criterion. If **forceSearch=true**, the field will instead display a search module field with auto-suggest functionality.

1.1.24.8 Custom Object Field Tag Attributes

size

Controls the width of custom object fields by specifying the number of characters for the field. For example:

Form
```
<TCFIELD NAME="ParentPolicy" size="10" />
```

Block
```
<tc:field name="ParentPolicy" category="EMPL" size="10"/>

<tc:wizardParameter name="ParentPolicy" size="10" />
```

display

Sets whether the field for selecting a related custom object appears as a search module or a drop-down list.

For example, if only a few records are available, displaying a drop-down list is sufficient. If many records might be available for the user to select, then the field should be displayed as a search module.

If a qualifier has been defined for the custom field, this qualifier applies no matter which display type is used. However, if a search view has been selected for the field, this search view is only used if the field appears as a search module. For details about these settings for a custom field of type Custom Object, see Custom Object Field Type.

The display type of custom fields of type Custom Object is initially determined by the default selection mechanism of the custom object, which is selected on the **General** tab of the custom object definition.

- Set the value to "`dropdownlist`" to make the field a drop-down list.
- Set the value to "`searchmodule`" to make the field a search module.

For example:

Form
```
<TCFIELD NAME="ParentPolicy" display="dropdownlist" />
```

Block
```
<tc:field name="ParentPolicy" category="CLAM" display="dropdownlist" />

<tc:wizardParameter name="ParentPolicy" display="dropdownlist" />
```

1.1.24.9 Involved Field Tag Attributes

size

Controls the width of Involved fields by specifying the number of characters for the field.

Note: *Among wizard parameter types, the type corresponding to custom fields of type Involved is known as Contact.*

For example:

Form
```
<TCFIELD NAME="Claimant" size="10" />
```

Block
```
<tc:field name="Claimant" category="CLAM" size="10" />
```

```
<tc:wizardParameter name="Claimant" size="10" />
```

1.1.25 Using Object Navigator

Object Navigator is a user interface tool you can use to specify a path to particular attributes in TeamConnect's object model for creating custom designs. End users generally do not have rights to Object Navigator or need to use it.

Many of TeamConnect design features require you to specify paths in the object model. For example, when defining a naming pattern for a custom object, you can use Object Navigator to specify the value of a field to be used in record names. Or you may need to specify a path when designing an EasyDocs document template.

TeamConnect's object model is comprised of attribute tables. Attributes tables are linked together to form a complex web of relationships. You can use Object Navigator to traverse these relationships and create a path that points to a specific field in the user interface.

To use Object Navigator, you need:

- An understanding of the object model in relation to the end-user interface
- Knowledge of attributes and paths
- A familiarity with Object Navigator and object models (Object Model: Read This First).

The following table describes where Object Navigator is located in TeamConnect.

Features That Use Object Navigator

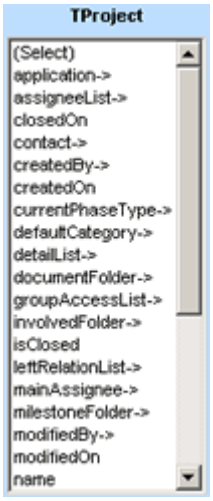
Feature	Tab	Option that uses Object Navigator
Conditions	General	Defining expressions for conditions that you save for all object definitions
Custom Object Definition	Name	Automatic naming of custom object records using a pattern of object attributes
Any Object Definition	Custom Fields	Creating a custom field of type Custom Object and defining a qualifier for that field

Rules	Qualifier	Defining qualifiers to specify the desired conditions that trigger a rule
	Action	Defining actions for Audit rules to specify a description of the triggering event or to capture data in custom fields
Routes	Stops	Selecting a **User With Role** or **User Path** to add as a stop member in a route
Search Views	Filter Display	Selecting object attributes to be used as search qualifiers
	Results Display	Selecting object attributes to be used as display keys
Templates	Fields	Assigning dynamic or static-dynamic values to fields with the help of the operators containing the "**value of path**" options
	Sub-objects	
	Related Objects	
Wizards	Page Actions	Assigning dynamic or static-dynamic values to fields with the help of the operators containing the "**value of path**" and "**value of parameter**" options

1.1.25.1 Attributes

Attributes are the building blocks of objects. Each attribute in the object model is mapped to a column in TeamConnect's database. Attributes act like placeholders for the actual data that users enter. Most attributes appear as fields in the user interface. For example, the **currentPhaseType** attribute is represented by a field labeled **Current Phase**.

Each object in TeamConnect consists of multiple attributes, most of which appear in the object's main table in Object Navigator. For example, attributes for custom objects appear in the **TProject** table.



The following items can be displayed in Object Navigator:

- **(End of Path) Attributes**—Identify data end values and do not link to other tables. When selected, they are the end of the path.

  For example, createdOn identifies the date a record was created.

- **Bridges**—Bridges are items you use to traverse (navigate the relationships) between tables in the object model to build paths. You cannot use them as an end of a path.

  Bridges always have an arrow (->) next to their name, and their names typically have the **List** suffix. For example, **assigneeList->** links to the **JProjAssignee** table, which lists attributes about project assignees.

  A bridge will only link to one other table. However, it is possible for multiple attributes to be bridges that link to the same table.

- **Bridge Attributes**—Bridge attributes are both attributes and bridges. Like bridges, they have an arrow (->) next to their name and can be used to traverse to another table. Because they also represent data, you can use bridge attributes to end a path.

For example, **defaultCategory->** identifies an object's default category, but it also links to the **WObjdCategory** table, which contains attributes that represent specific information about the category.

For a complete list of TeamConnect attributes, including which attributes can be used to traverse tables, end paths, or both, see Object Model: Read This First.

1.1.25.2 Paths

A path is a specific route to the location of an attribute in the object model. It is a sequence of object attributes separated by periods (.) that identifies the location of the needed value. For example: .mainAssignee.type is a path that identifies the role of the project's main assignee.

Depending on the starting object table, you can distinguish two types of paths:

- **Simple**—An attribute in the current object table (the starting table) without traversing other tables.

  For example, if you want a rule to check which user created a record, you could select the **createdBy->** bridge attribute in the current object table without traversing to another table. TeamConnect follows the resulting .createdBy path to find the value.

- **Complex**—A sequence of attributes that traverse various object tables, using the relationships between them (bridges and bridge attributes). If the required value resides in a different table, you must build a complex path to it.

  For example, if you want to include the plaintiff's name in the record naming pattern of the custom object Litigation, you could navigate to **Plaintiff**, a custom field of type Involved:

  detailList-> Litigation-> Plaintiff

  The resulting path appears as:

  ```
  .detailList(Litigation).detailInvlValueList(Plaintiff).detailValue.contact.
  ```

After you have used Object Navigator to create a path to an attribute, the path appears as text in the field associated to the **Object Navigator** icon .

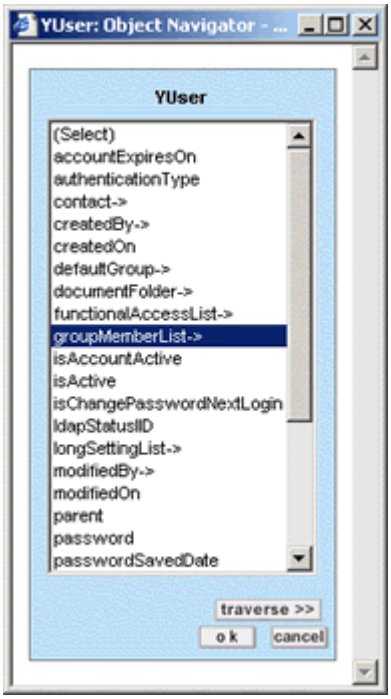
1.1.25.3 Object Navigator Window

You can use Object Navigator with Designer screens that contain a field for identifying an attribute in the object model. These fields display the **Object Navigator** icon .

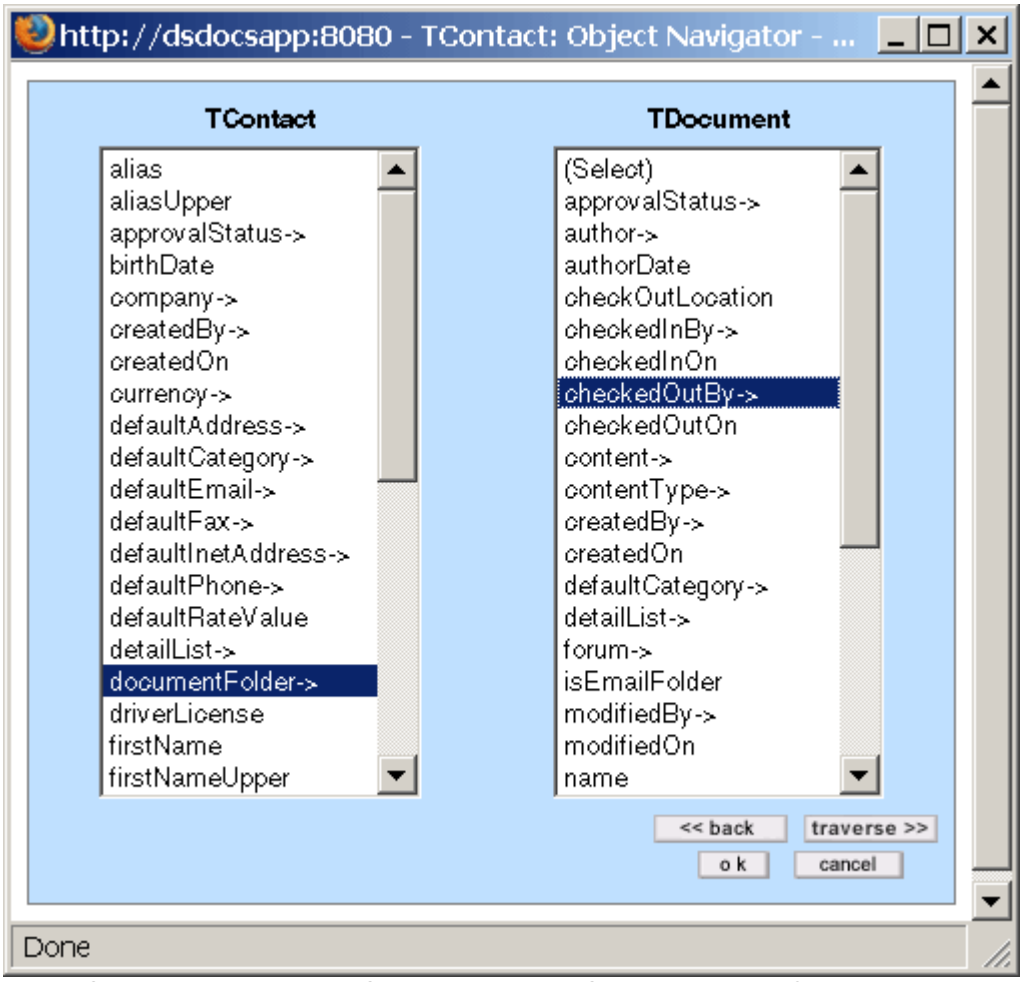
Some screens always display the **Object Navigator** icon, such as the **Qualifier** tab in rule screens. Other screens only display the **Object Navigator** icon when certain selections are made. For example, the **Fields** section of the **Templates** screen only displays the **Object Navigator** icon when particular operators are selected, such as **Attribute** in the **Value** field.

To start Object Navigator, click the **Object Navigator** icon next to the field for which you want to define an attribute path. Object Navigator appears in a separate window that displays the starting table in the object model from which your path will originate. You can create a simple path by selecting an attribute in the starting table or a complex path by selecting a bridge or bridge attribute and traversing to open the next table. For details, see Creating Paths.

Object Navigator displays the active table on the right, each table you have traversed on the left, and highlights each attribute that you add to the path.



Object Navigator Window with Only a Starting Table



Object Navigator Window Showing Traversed Starting Table and Current Table

The following table describes the items in the **Object Navigator** window.

Object Navigator Window

Item	Description
Table name	Name of traversed tables or current table. The table names of system objects appear as they are defined in the object model, such as **TAppointment**. The starting tables of custom objects are named after the custom object, such as **Dispute**.
Current table	Displays the available attributes and bridges in the currently selected table. Select an attribute or bridge to add it to your path.
Traversed table	Displays a traversed table with the selected bridge or bridge attribute highlighted in the table.
traverse	First select a bridge or a bridge attribute and then click **traverse** to:

	• Add the bridge or bridge attribute to your path • Open the table to which the bridge points
back	Click to go back to the previous table. When you click **back**, the selection is removed from the path. This button is only displayed after you have traversed at least one table. You can click **back** and **traverse** as needed to select different attributes or to traverse to different tables.
ok	After you have selected your final attribute, click **ok** to complete the path.
cancel	Click to close Object Navigator and cancel the path.

1.1.25.3.1 Starting Tables

The first table that you see when you open Object Navigator is the starting table, which is the point from which your path originates.

Specifying your starting table is important because it corresponds to a table in the object model. Object Model: Read This First explains each table's attributes and where any bridges and bridge attributes lead. Using this information about your starting table, you can make selections that navigate to the desired attribute.

The starting table typically depends on the object for which you are building the path. For example, if you are defining rules, templates, search views, or wizards for an object, the starting table is the main table of that object (usually with the prefix **T**). There are 'T" tables for eleven system objects, including Involved (for example, **TInvolved**).

For custom objects, the starting table is **TProject**. The **TProject** starting table is labeled with the name of the custom object, such as **Matter** or **Claim**, unless the specific custom object is not known. If you are building a path for a current user, the starting table is **YUser**.

The following table provides a list of starting tables that appear in Object Navigator in different screens. If you are not sure which attribute, bridge, or bridge attribute to select from these tables, refer to Object Model: Read This First.

Object Navigator Starting Tables

Screen	Tab	Object drop-down list options	Starting table
Conditions	**General**	**Current Object**	T-table of the object you are working with, such as **TAccount**.
		Current User	**YUser**

Custom Object Definition	**Name**	N/A	**TProject** (labeled with the name of the current custom object for which you are creating the naming pattern, such as Dispute).
Any Object Definition	**Custom Fields**, for custom fields of type Custom Object	**this.**	**TProject** (labeled with the name of the custom object selected for the field's definition).
		container.	T-table of the object you are working with, such as **TAppointment**.
Rules	**Qualifiers**	**Current Object**	T-table of the object you are working with, such as **TAccount**.
		Current User	**YUser**
Routes	**Stops**	N/A	T-table of the object with which the route is associated, such as **TTask**.
Templates	**Fields**	N/A	T-table of the object you are working with, such as **TTask**.
	Sub-Objects	N/A	J-table of the sub-object, except categories, such as **JContAddress** EDetail table for categories, such as **EContDetail**.
	Related Objects	N/A	T-table of the related object you are adding to the template, such as **TAccount**.
Wizards	**Page Actions**	**Current Object**	T-table of the object you are working with, such as **TAccount**. **TProject** (labeled with the name of the custom object) is the starting table for custom objects. In Object Navigator, it is always displayed with the name of the

			custom object, such as **Case**, **Matter**, and **Policy**, except in cases where it is representing all available custom objects.
		Related Objects (from template)	T-table of the related object added to the associated template, such as **TAppointment**.
		Sub-Objects (from template)	J-table of the sub-object, such as **JProjAssignee**.
	Qualifiers (accessed by **Page Transitions**) **Parameters** of type:	Lookup table items	The corresponding L-table of the item, such as **LContPhoneType**.
		Category	**WObjdCategory**
		Custom object	**TProject** (labeled with the name of the custom object for which you are creating the wizard).
		User	**YUser**
		Group	**YGroup**
		Account	**TAccount**
		Contact	**TContact**

1.1.25.3.2 Filtering Sub-objects and Related Objects

If an attribute links to a list of sub-objects or related objects, you have the option of incorporating a filter into the path. The filter could be the sub-object's role or related object's category. For example, you may want to filter matter assignees by role or involved parties by category (role). This is done with the use of a special intermediary table that appears in Object Navigator but is not actually part of the object model.

In this intermediary table between the starting table and the related or sub-object table, you can do either of the following:

- Choose the specific item for the filter.
- Select (Any)->, meaning that you do not want the list to be filtered.

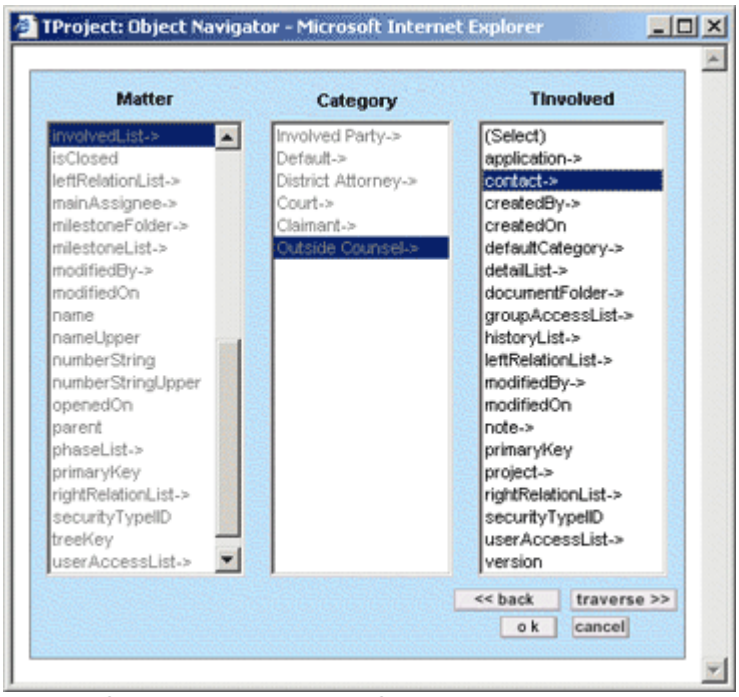
For example, to create a path from a matter to its involved parties' contact records which have the role Outside Counsel, you could specify the following:

involvedList-> Outside Counsel-> contact-> (ok)

This would result in the following path:

```
.involvedList(Outside Counsel).contact
```

The following image shows what this path looks like in Object Navigator:



Object Navigator Related Object Filtering Example

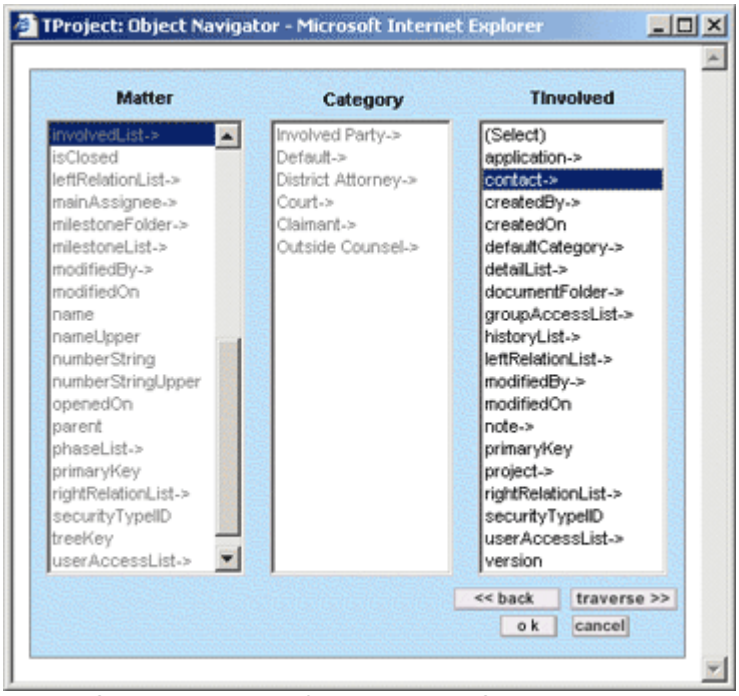
If you want to create a path from a matter to its involved parties without filtering them by their role, you could make the following selections:

involvedList-> (Any)-> contact-> (ok)

This would result in the following path:

```
.involvedList(Any).contact
```

The following image shows what this path looks like in Object Navigator:



Object Navigator Unfiltered Related Object Example

The third table, which represents the involved parties, is the same whether you filter the list with an involved role or not. From the TInvolved table, you can access the contact information for the involved parties, such as their phone numbers or rates, through the contact-> attribute.

The following table indicates which attributes lead to intermediary, filtering tables in Object Navigator and the table to which the attribute links in the object model.

Object Navigator Attributes That Link to Intermediary Tables

Object table	Attribute	Leads to this intermediate table in the user interface	Then leads to this table in the object model
TProject	`accountList->`	**Category**	**TAccount**
	`appointmentList->`	**Category**	**TAppointment**
	`assigneeList->`	**Assignee Role**	**JProjAssignee**
	`childList->`	**Child Object**	**TProject** (labeled with the custom object's name)
	`expenseList->`	**Category**	**TExpense**
	`involvedList->`	**Involved Role**	**TInvolved**

	taskList->	**Category**	**TTask**
	rightRelationList->	**Relation Type**	**JProjRelation**
	leftRelationList->	**Relation Type**	**JProjRelation**
TContact	addressList->	**Address Type**	**JContAddress**
	emailList->	**Email Type**	**JContEmail**
	faxList->	**Fax Type**	**JContFax**
	phoneList->	**Phone Type**	**JContPhone**
	inetAddressList->	**Internet Address Type**	**JContInetAddress**
	skillList->	**Skill Type**	**JContSkill**
	rateList->	**Task Category**	**JContRate**
	rightRelationList->	**Relation Type**	**JContRelation**
	leftRelationList->	**Relation Type**	**JContRelation**
TInvolved	rightRelationList->	**Relation Type**	**JInvlRelation**
	leftRelationList->	**Relation Type**	**JInvlRelation**
TAppointment	resourceList->	**Resource Type**	**JApptResource**

1.1.25.3.3 Creating Paths

Before you create a path with Object Navigator, make sure that you are familiar with the following:

- Object Navigator and its components
- The concept of your starting table
- The location of the end of path attribute that you need
- Attributes, bridges, and bridge attributes the path requires

For details about starting tables, attributes, and the TeamConnect object model, see the Object Model: Read This First.

- Necessary settings in the screen where you are creating a path (for example, wizards, templates and rules)

Tip: *If you know the attributes to include in the path, you can type the path into the field. Separate each attribute with a period (.)*

To create a path with Object Navigator

1. In the screen where you want to create a path, click the **Object Navigator** icon.

   Object Navigator opens, displaying the starting table.

2. If you know how to reach the necessary attribute, build the path and go to step 6.
3. If you are not sure which attribute to select, find the your starting table in Object Model: Read This First.

   Tip: *For custom objects, look up the* ***TProject*** *table in Object Model: Read This First.*

4. Look through the **Comments** column of the selected table in Object Model: Read This First and determine whether there is an attribute in the current object table that contains the value you need. If you find the necessary attribute or bridge attribute in current table, select it in Object Navigator and go to step 6.
5. If you do not find the attribute in the current table:
   a. Find the bridge that can link you to the appropriate table. If necessary, look up the tables to which the bridges link in Object Model: Read This First to see if they contain the attribute you need.
   b. Select the bridge and click **traverse**.
   c. A new table appears to the right of the starting table.
   d. Repeat steps 2 through 4 until you have navigated to the necessary attribute.

   Tip: *If you have traversed more than three tables but have not found the necessary attribute, you might be on the wrong track. Click* ***cancel*** *and start over.*

6. Click **ok**.

   Object Navigator closes and the path that you have created appears in the field.

7. To complete the settings for the path, make other selections required in the screen you are using.

   The path is completed and displayed in the field. You can click the Object Navigator icon again and edit the path if necessary. Object Navigator opens, returning to the table where you left off. You can then select a different attribute, click back, or traverse to a different table.

1.1.25.4 Tips for Navigating Objects

This section provides tips for creating paths with Object Navigator and lists examples of common bridges and paths.

Practical Tips for Building Paths

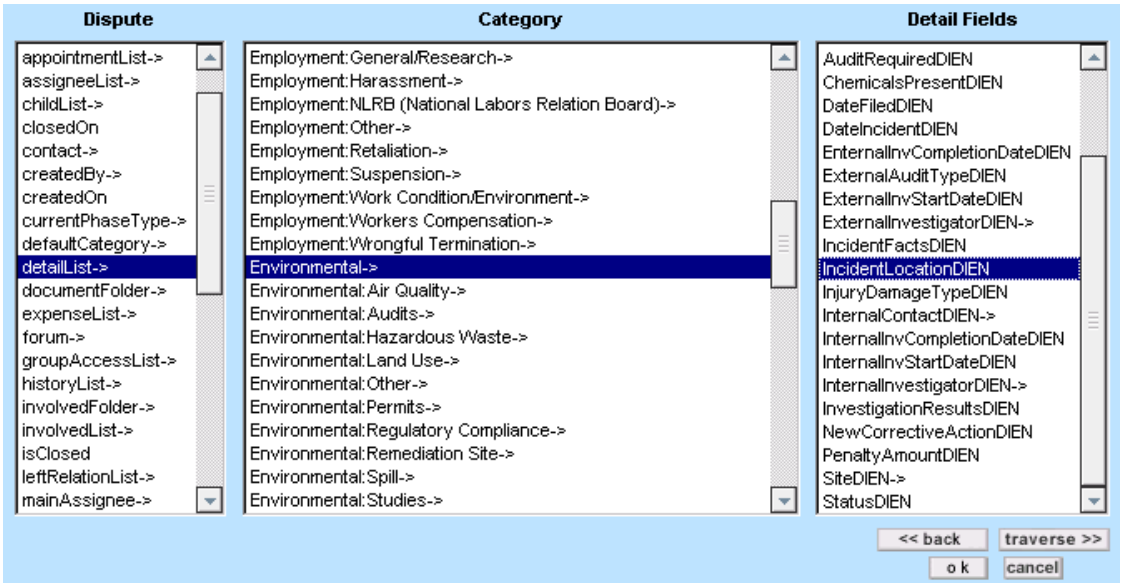
- The field label can help you determine which attribute to select. For example, the currentPhaseType attribute corresponds to the Current Phase field. However, there are two exceptions to this general rule:
  - The type-> attribute (instead of "role") represents project assignee roles in the JProjAssignee table.
  - The owner-> attribute (instead of "task" or "project") for task or project assignees (in the JProjAssignee and JTaskAssignee tables) identifies assignees for a task or project.
- To create a path, you typically do not have to traverse more than one or two tables. If you have traversed over three tables and still cannot find the required attribute:
  - You may have selected the wrong bridge. If so, click back or cancel and start again. For assistance, check Object Model: Read This First.
  - The required attribute may not be accessible from the current object table. If so, you may have to write a Java action for the desired operation.
- When constructing a path, make sure that the end of path value type matches the value type of the field whose value you are setting.

  For example, if you are setting a value for a number field and the path ends with a date, you cannot add the path. Or, if you are building a path that identifies a user, the end of path attribute must be in the YUser table.
- In most cases, displayOrder, version, and primaryKey are not useful.
- In certain screens, attributes with the suffix "List" cannot be traversed, so they are not displayed with an arrow (->). This indicates that the related table is not applicable.

Common Bridges and Bridge Attributes

- The bridge attribute contact-> in the TProject table represents the contact selected for contact-centric custom objects.
- Items that have the "List" suffix are bridges that typically link to sub-objects, such as a project's assignees.
- To specify a user group, follow the .groupMemberList.group path from the YUser table.
- If you need to identify a field containing a system lookup table item, such as the field containing the role of a project assignee, your last selection in the path should be the bridge attribute type->. For example:
  - .assigneeList.type
  - .rightRelationList.type
  - .addressList.type

- Custom fields created for an object can be reached by traversing the bridge detailList-> to the Category table, which links you to the list of all custom fields created for the selected category as shown in the following example:



Example of Traversing to Custom Fields

See also the example of a complex path in Paths.

Commonly Used Paths

The following table lists the origins and destinations of common paths and the selections in Object Navigator used to build them. For more information on specific object model tables and attributes, see Object Model: Read This First.

Common Object Navigator Paths

From this object	To this attribute	Path
Any **T-table** (any object)	Custom fields of the current object	1 detailList-> (traverse) 2 categoryName-> (traverse) 3 customFieldName (ok)
	A record's default category	**defaultCategory->** (ok)
	Any record category	1 detailList-> (traverse) 2 categoryName-> (ok)
	The main assignee of a project that is selected in a custom field of type Custom Object	1 detailList-> (traverse) 2 categoryName-> (traverse)

		3 detailFieldName-> (traverse) 4 mainAssignee-> (traverse) 5 user-> (ok)
	The user who created a record	**createdBy->** (ok)
TExpense	The total amount of an expense	**totalAmount** (ok)
	The main assignee of a related project	1 project-> (traverse) 2 mainAssignee-> (traverse) 3 user-> (ok)
TInvoice	The main assignee of a project identified in an invoice line item	1 lineItemList-> (traverse) 2 project-> (traverse) 3 mainAssignee-> (traverse) 4 user-> (ok)
	The net total of a line item	1 lineItemList-> (traverse) 2 netTotal (ok)
	The task category of a line item	1 lineItemList-> (traverse) 2 taskCategory-> (ok)
	The expense category of a line item	1 lineItemList-> (traverse) 2 expenseCategory-> (ok)
	The category of a vendor	1 vendor-> (traverse) 2 detailList-> (traverse) 3 categoryName-> (ok)
	The date of the invoice	**invoiceDate** (ok)
TProject (any custom object)	An assignee with a specific role	1 assigneeList-> (traverse) 2 <role>-> (ok)
	A user assigned to a project with a specific role	1 assigneeList-> (traverse)

		2 <role>-> (traverse) 3 user-> (ok)
	The current phase of a project	**currentPhaseType->** (ok)
	The user who is the project's main assignee	1 mainAssignee-> (traverse) 2 user-> (ok)
	The role of a project's main assignee	1 mainAssignee-> (traverse) 2 type-> (ok)
	The main contact of a contact-centric project	**contact->** (ok) ***Note:*** *Only available when the custom object is contact-centric.*
	Custom fields of a parent project of the current object	1 parent-> (traverse) ***Note:*** *Only available when there is a parent-child relationship.* 2 detailList-> (traverse) 3 categoryName-> (traverse) 4 customFieldName (ok)
YUser	The user's group membership	1 groupMemberList-> (traverse) 2 group-> (ok)
	The user's default group	**defaultGroup->** (ok)

1.1.26 Object Model: Read This First

This appendix introduces a series of appendixes with reference tables to aid in the use of the TeamConnect object model. These tables provide information about objects, attributes and relationships in TeamConnect's object model, which you can use to create custom designs.

These appendixes do not describe the objects that construct the administrative and design functionality in TeamConnect, or TeamConnect's database structure. There are many parallels between the database structure and the object model, but they are distinct.

The appendixes are:

- Object Model: Projects
- Object Model: Contacts
- Object Model: Appointments
- Object Model: Accounts
- Object Model: Expenses
- Object Model: Tasks
- Object Model: Invoices
- Object Model: Histories
- Object Model: Documents
- Object Model: Common

1.1.26.1 Objects and Object Model

Each table in these appendixes corresponds to an object in the object model. You may use these tables to understand the structure that organizes data in TeamConnect. For example, use these tables when you are doing any of the following actions:

- Using TeamConnect's Object Navigator tool in the user interface while creating rules, templates, wizards, wizard page transition rules, or automated object naming patterns based on attribute values.
- Creating templates using Document Generator or EasyDocs.
- Selecting attributes from a list (rather than from Object Navigator), such as when creating qualifiers in a search view or setting an object's Unique ID to a pattern of certain attributes.

1.1.26.2 Legacy Names and Newer Names

Some attribute names were changed in TeamConnect version 3.4. The new names are used when developing custom blocks with XML. The legacy names are used when working with search views, rules, Object Navigator, and other TeamConnect features. When an attribute has both a legacy name and a newer name, that fact is noted in its description.

1.1.26.3 Intended Audience

These appendixes are primarily intended for solution developers who are involved in the process of customizing TeamConnect to meet the needs of an organization. The user of these appendixes is expected to have the following skills and knowledge:

- Conceptual understanding of:
  - Object-oriented programming
  - Relational databases
- Analytical skills

- Understanding of the business needs of the organization for which you are developing a solution

1.1.26.4 Object Model Conventions

- Each table in this documentation includes all of the attributes available in Object Navigator for the corresponding object. This includes relationships that do not exist in the database.
- Attributes are listed in alphabetical order within each table.
- You can use the grouping of tables in the left pane to understand the relationships between the various tables. The tables are grouped in the left pane according to the main objects to which they are most closely associated (such as Account, Project, or Task). Tables that are administrative and are not closely associated with one particular object are listed in the **Common** section.
- Each main object's tables are listed in the following order:
  a. T-table
  b. J-tables and their corresponding R-tables
  c. Category and Custom Field tables (E-tables)
  d. User and group access tables
- Object (table) names are provided as links, so that if you are viewing this documentation electronically, you can quickly navigate to the table for the corresponding object.
- Items that have the data type **object** are treated as objects, rather than foreign keys (as they are treated in the database) in the descriptions that are listed in the **Comments** column of each table. This is why the **Comments** column does not indicate that an attribute stores the ID or primary key of the object, but instead indicates that the attribute identifies the object itself. This convention is followed in order to lessen confusion for users who are not concerned with the way the database actually works.

  For example, in the table **TInvolved**, the attribute **project-->** is described as the project to which the involved record belongs, rather than the primary key of the project to which the involved record belongs.

1.1.26.4.1 Enumerations

Enumerations are represented in the Object Model by `typeIID` (Internal ID) attributes that can only have static, predefined values. The predefined values of these enumerations are listed in parentheses in the **Comments** column of the table for each such attribute. In some queries you may wish to use the integer that represents each value. In other situations, such as embedding report tags into custom blocks, you will need to use the Java constant name for the value instead of the integer.

As an example, consider attribute `typeIID` in entity `TInvoice`. It has four possible values:

Integer	Java constant	Text shown to end user (US English locale)
0	STANDARD	Standard

1	CREDIT_NOTE	Credit Note
2	ACCRUAL	Accrual
3	SHADOW	Shadow Invoice

The text displayed to the end user for an enumeration may vary depending upon the locale that is being used for the session. However, the Java constant names will always be those described in the Comments for the attribute, regardless of locale.

If you are querying the actual database value, you would look for one of the integers (0 through 3). If you are referencing this attribute in connection with a report tag in a custom block you would use the Java constant name that corresponds to the value you are looking for.

The datatype shown for enumerations in the Object Model tables is **Enum**. This usually represents an integer datatype in the underlying database (although some enumerations are character datatypes), extended with predefined values and Java constant names.

Note: *All enumerations in this Object Model have attribute names ending in "IID". There is further information about enumerations in .XCT files that are provided in the installation media; see Viewing XCT Files.*

1.1.26.4.2 Object Model Terms

You must understand the meaning of the following terms in order to work with this document:

- **Object** is used to refer to objects in the object model. This is a different definition for this term than for other TeamConnect documentation. The distinction is that in TeamConnect's GUI, only the main objects are treated as objects (for example, System Objects, Custom Objects, Related Objects, and so on). In the object model, this term is more broad.
- **Object Navigator** is a UI tool that allows you to navigate through tables that represent objects in the object model. This makes it easier for you to select attributes when necessary for the task you are completing. For more details about Object Navigator, see Using Object Navigator.
- **Attributes** are the building blocks of objects, and represent the actual data that end user enter into TeamConnect. They also represent relationships between objects. For example, every Custom Object consists of over a dozen attributes, all of which are listed in the main object table for Project: mainAssignee, createdOn, createdBy, closedOn, defaultCategory, and so on. All of the attributes for an object are listed in its corresponding table in the document.
- **End of path** If an attribute is an **end of path**, it can be selected at the end of a path in Object Navigator. An end of path contains some type of value, such as a number, date, or text.

  If an item cannot be selected as the ending of a path, it is a "bridge" and exists in the object model for the purpose of establishing relationships to other objects.
- **Bridge** If an attribute is a Bridge, it allows you to traverse to other tables in Object Navigator that correspond to objects in the object model. Some bridges correspond to database columns and represent a specific data value. However, not all bridge attributes exist in the database. If an item is marked as a bridge but not an end of path, it does not represent a database column. It

represents a one-to-many relationship and exists in the object model for the purpose of establishing relationships to other tables, and can never be selected as an end of path.

- **Table** is used to refer to objects, for three reasons:
  - Each table in this document represents an object in the object model. This format is used to convey the information in a logical and visual manner.
  - Object Navigator represents objects by displaying them as tables in the UI.
  - All of the objects in this document have corresponding tables in the database, with corresponding columns for most attributes.

1.1.26.4.3 Descriptions of Columns in Tables

The columns in each table in this document are provided to give you a good understanding of each attribute. The following columns are included in each table:

TeamConnect Attribute	Name of the attribute as it appears in Object Navigator.
Field in UI	System field in TeamConnect's end-user UI (as provided by default with the system) that corresponds to the attribute. The location of the field in the UI is indicated in parentheses. Not all attributes are represented by fields in the UI. This document does not attempt to reflect any customization of System Fields in the UI that has been performed for your system.
End of path	Indicates whether the item can be selected at the end of a path in Object Navigator. See Object Model Terms for details.
Bridge	Indicates whether the item is a bridge. This is also indicated by an arrow (-->) at the end of the attribute name. See Object Model Terms for details.
Links to object table	If the item is a bridge, this column indicates the object in the object model (or, the table in Object Navigator tool) to which the bridge provides a relationship. ***Note:*** *If the item is ALSO an attribute, then the primary key of this object is the value that the attribute stores in the database.* ***Tip:*** *When viewing this document electronically, you can click the name of the table to go directly to that object's attribute descriptions in this document.*
Data type	Indicates the data type of the attribute.

	• If the data type is **string**, the character limit is indicated in parentheses (). • If the data type is **longString**, the limit is imposed by the database. In Oracle, the maximum size of a longString value is 4 GB. In SQL, the maximum size is 1 GB. • If the data type is **datetime**, there is an additional property to consider. Some datetime attributes are **time-zone-independent**. This means that the value in the database for this attribute is retrieved and displayed as is, regardless of the local time zone of the user who entered that value, or the time zone of the user who is now viewing that value. Other attributes are time-zone-dependent. Their value is stored in the database but is adjusted during retrieval to compensate for the time zone of the user who is viewing the value. In this documentation, the phrase "time-zone-independent" will appear for attributes that possess this property. Custom datetime fields that are created by solution developers are configurable. The solution developer can choose whether or not to make the field time-zone-independent. • Bridges are complex data types, and are marked in this documentation with the data type **object**, except when the bridge links to a list of records rather than a single record.
Comments	Provides a description of the item. For beginners, this column may also provide suggestions for using the item. For items that are marked **End of path** and **Bridge**, two bulleted descriptions are given in this column. These bullets describe the two ways that you can use this item. For example: **8** The first item is marked with this bullet and describes the item's function when selected as an end of path. For example, selecting user--> as the end of your path in the table JProjAssignee gives you the user who is being identified as an assignee of the project. **-->** The second item is marked with this bullet and describes the contents of the table to which the item links, if you use the item as a bridge. For example, selecting user--> as a bridge in the table JProjAssignee gives you access to the table YUser, which contains that user account's definition information. These two descriptions are provided in order to help you determine the following: • Whether or not the item should be included as part of your path. • Whether the item is the attribute you are trying to reach and should therefore be selected as the end of your path.

	• Whether you need to use the item as a bridge, and link to the table to which the bridge provides access.
Commonly used in (Rules, Templates/ Wizards)	These columns have been introduced primarily to help you learn to build a path using Object Navigator. Because Object Navigator displays all existing attributes, regardless of their function, there are many selections available that serve no practical purpose when you are creating Rules, Templates, Wizards, and Wizard Rules. You can use these columns to help you distinguish between useful and non-useful selections. For example, in most cases, the Version attribute is not considered a commonly used attribute for creating Rules, Templates, or Wizards.

1.1.26.5 Object Model Properties

The organization of the object model follows several conventions. Becoming familiar with these conventions will help you understand the structure of the object model. The following sections also contain reference material, such as the unique codes of TeamConnect objects.

1.1.26.5.1 Object Table Prefixes

Object tables are named with a one-letter prefix according to their function in the object model. The table below indicates the various types of tables and their prefixes.

Prefix	Description	Examples
T	T-tables are the 11 main tables for TeamConnect objects. In Object Navigator, they are almost always the starting point of navigation. A T-table contains the basic information for each record of its type that is created. In other words, it contains instances of an object definition. T-tables own specific instances of custom fields (E-tables), categories (E-tables), and sub-objects (J-tables), all of which store items added to the record in batch screens. T-tables are not owned by any other kind of table. However, a T-table might have a relationship to another T-table. For example, T-History is related to all of the other T-tables, because History records can be created for each kind of record.	• **TAccount** • **TAppointment** • **TContact** • **TDocument** • **TExpense** • **THistory** • **TInvoice** • **TInvolved** • **TProject** • **TTask**
J	J-tables are owned by T-tables (or, occasionally, R-tables). J-tables are also	• **JApptAttendee** • **JContAddress**

	known as sub-objects. The values stored in J-tables are usually obtained from lookup tables (L-tables). In the UI, items in a J-table are added to a record through a batch screen. For example, assignees added to a project in a batch screen are sub-objects of the project.	• **JContEmail** • **JProjAssignee** • **JTranDetail**
L	L-tables store the main lookup table information for system lookup tables.	• **LContFaxType** • **LCountryItem** • **LProjAssigneeType** • **LContSkillType**
E	E-tables are used with every T-table throughout the object model to store the following: • specific categories that have been added to records • specific custom field values added to records • specific user or group security settings for records	• **EHistDetail** (contains categories that have been added to History records) • **EHistDetailNumbValue** (contains values that have been added in custom fields of the type Number, in History records) • **EHistUserAccess** (contains user security settings for History records)
R	There are only four R-tables in the object model. Each of the R-tables has a unique purpose (shown at right). Like T-tables, R-tables are not owned by any other table. However, they do not contain all of the attributes found in a T-table. For example, they do not have custom fields or categories associated with them, and they do not belong to an object definition.	• **RTransaction** (stores instances of financial transactions) • **RDistribution** (stores contact groups, that is, Address Books) • **RApproval** (stores instances of approvals) • **RForum** (stores any forums created for records)
Y*	Y-tables are system tables. They typically contain values that correspond to	• **YGroup** • **YUser**

	administration objects, such as groups and users. They are owned by Z-tables. Custom lookup table information is also stored in Y-tables.	• **YDetailLookupTable** • **YDetailLookupItem**
W*	W-tables store object definition information. The components related to object definition are stored in these tables. They are owned by Z-tables. Categories and custom field definitions for all objects are also stored in W-tables.	• **WObjdCategory** • **WObjdDetailField**
Z*	Z-tables own W-tables and Y-tables. They contain specific information pertaining to system design, such as object, template, wizard, and search view definition information.	**ZObjectDefinition**
U*	U-tables are administrative tables that are owned by W-tables.	**UGrupMember**
* Most tables that are named with this prefix are not typically used when creating rules, templates, wizards, or object naming patterns, although there are certain exceptions. For example, **UGrupMember** is used in rules that check whether the current user is a member of a certain group.		

1.1.26.5.2 Unique Codes of System Objects and Lookup Tables

Each system object and each system lookup table has a four-character unique code. You will need to use these unique codes for certain tasks such as writing document template for Document Generator or writing automated qualifiers or actions for rules.

The following table lists the unique codes and database table names for system objects.

System Object	Object Table Name	Database Table Name	Unique Code
Account	TAccount	T_ACCOUNT	ACCT
Appointment	TAppointment	T_APPOINTMENT	APPT
Contact	TContact	T_CONTACT	CONT
Document	TDocument	T_DOCUMENT	DOCU
Expense	TExpense	T_EXPENSE	EXPE

History	THistory	T_HISTORY	HIST
Invoice	TInvoice	T_INVOICE	INVC
Invoice Line Item	JInvcLineItem	J_INVC_LINE_ITEM	LNI$
Task	TTask	T_TASK	TASK

The following table lists the unique codes and database table names for all system lookup tables.

System Lookup Table	Object Table Name	Database Table Name	Unique Code
Activity Item	LTaskActivityItem	L_TASK_ACTIVITY_ITEM	ACTI
Address Type	LContAddressType	L_CONT_ADDRESS_TYPE	ADDR
Area Item	LApptAreaItem	L_APPT_AREA_ITEM	AREA
Contact Relation Type	LContRelationType	L_CONT_RELATION_TYPE	CONR
Country Item	LCountryItem	L_COUNTRY_ITEM	COUN
Currency Item	LCurrencyItem	L_CURRENCY_ITEM	CURR
Email Type	LContEmailType	L_CONT_EMAIL_TYPE	MAIL
Fax Type	LContFaxType	L_CONT_FAX_TYPE	FAXX
Internet Address Type	LContInetAddressType	L_CONT_INET_ADDRESS_TYPE	INET
Phone Type	LContPhoneType	L_CONT_PHONE_TYPE	PHON
Project Assignee Type	LProjAssigneeType	L_PROJ_ASSIGNEE_TYPE	Unique code of the corresponding custom object definition.
Project Relation Type	LProjRelationType	L_PROJ_RELATION_TYPE	PRJR

Resource Type	LApptResourceType	L_APPT_RESOURCE_TYPE	RESO
Skill Type	LContSkillType	L_CONT_SKILL_TYPE	SKIL
Territory Type	LContTerritoryType	L_CONT_TERRITORY_TYPE	TERR

1.1.26.5.3 Viewing XCT Files

To verify information about the attributes in any table, such as the data type or equivalent database column or table name, you can view the XCT file corresponding to the table. The XCT files provide mapping between the database and the object model. These files are used by Object Navigator to obtain the correct attributes. In the XCT files, the external name of an attribute or table is its name in the TeamConnect database.

The XCT files are located on the TeamConnect installation media in the **utilities/objectmodel** directory. This directory contains two folders: **simpletypes** and **complextypes**.

- **utilities/objectmodel/complextypes** contains the XCT files for all objects in the object model.
- **utilities/objectmodel/simpletypes** contains the XCT files for `typeIID` attributes. These object attributes are enumerations that are permitted to have only certain static values that have been predefined. Each file in the **simpletypes** directory provides a list of possible values for the corresponding `typeIID` attribute. The values are also listed in this document in the **Comments** column for each `typeIID` attribute.

1.1.26.5.4 Object Names vs. Database Table Names

The names of most tables in the object model are similar to their corresponding table names in the database. Also, the names of most attributes in the object model are similar to their corresponding column names in the database. The name of a table or column in the database is usually the name of the table (or attribute) in the object model, except that it is in caps, with underscores between words. For example:

LContRelationType = L_CONT_RELATION_TYPE

shortDescription = SHORT_DESCRIPTION

TAccount = T_ACCOUNT

This convention is used for most, but not all, tables and attributes. To verify the name of a table or column in the database, refer to the corresponding XCT file, and check the External Name of the attribute or table, as described in Viewing XCT Files.

1.1.26.5.5 List Attributes for Sub-Objects and Related Objects

Many bridge attributes have the suffix **List**. In Object Navigator, bridge attributes appear with an arrow after the suffix, like this: **List-->** This suffix is added to attributes that link to sub-objects that are added to records (for example, JContAddress, JContPhone, and JProjAssignee) and related objects that are added to records (for example, THistory, TInvolved, and TTask).

The following are important points about attributes with the suffix **List**:

- When you want to access sub-objects that have been added to a record, you should use the corresponding J-table. This includes contact addresses, contact phone numbers, appointment attendees, project assignees, and other items that are added in batch screens.

  The one exception to this rule is categories, which are accessed through **detailList-->** attributes. Although they are added in a batch screen in a record and are handled as sub-objects, they are stored in an E-table rather than a J-table.

- When you want to access related objects that have been added to a record, such as the involved records added to a project, you should use the List attribute that corresponds to it (in this case, **involvedList-->**) to access the corresponding T-table (TInvolved).

- In Object Navigator, most attributes with the suffix **List** link first to an intermediary table that allows you to filter the related or sub-objects.

1.1.27 Object Model: Projects

The tables in this appendix provide information about the object model as it relates to projects.

1.1.27.1 TProject

TProject contains the general information for a project record. This information includes the project's name, creation date, closing date, parent project, main assignee, and other information. This table also links to the list of a project's assignees, which are sub-objects of a project, categories added to the project, and related objects of the project (such as child projects and involved records).

Object: TProject (T_PROJECT)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
accountList-->	**n/a**	**Accounts** tab	**x**	**x**	TAccount		The list of accounts that have this project selected in their posting criteria. Links to account information for	**x**	

							the related accounts. In Object Navigator, links to an intermediary table named **Category** that allows you to filter the accounts based on a category, if desired.		
application-->	**APPLICATION_ID** **(NUMBER)**		**x**	**x**	WObjdProjectInfo	object	The Custom Object definition to which this project belongs.		
appointmentList-->	**n/a**	**Appointments** tab	**x**	**x**	TAppointment		8 The list of appointments that are related to the project. 8 Links to specific appointment information. 8 In Object Navigator, links to a table named **Category** that allows you to filter the appointments based on a category, if desired.		
assignees--> **Before TeamConnect 3.3**	**n/a**	**Assignees** tab	**x**	**x**	JProjAssignee		8 The list of assignees added to the project.	**x**	

SP2: assigneeList-->							--> Links to the join table that contains specific information for assignees added to the project. In Object Navigator, links to a table named **Assignee Role** that allows you to filter the assignees based on a role, if desired.		
embeddedProjects--> **Before TeamConnect 3.3 SP2: childList-->**	**n/a**	Tabs for child or embedded object records	**x**	**x**	TProject		8 The list of all child projects of the current project, including child and embedded objects. Does not include grandchild objects. --> In Object Navigator, links to the **Child Object** table, which allows you to select the child or embedded custom object whose records you want to identify in your path. After you select a child object, links to the **TProject** table.	**x**	

							The **Child Object** table is not an object in the object model and appears only in Object Navigator to allow you to identify which child object's records you need.		
closedOn	**CLOSED_ON** **(DATE)**	**Closed On** (**General** block)	**x**			date	Date the project status was changed to Closed.	**x**	
contact-->	**CONTACT_ID** **(NUMBER)**	Field for selecting a contact, with customized label, in **General** block	**x**	**x**	TContact	object	8 The contact selected for a contact-centric project. --> Links to the specific information about the contact object.		**x**
createdBy-->	**CREATED_BY_ID** **(NUMBER)**	**Created By** (**Security** block)	**x**	**x**	YUser	object	8 User who created the record. --> Links to specific information about the user account.	**x**	**x**
createdOn	**CREATED_ON** **(DATE)**	**Created On**	**x**			date	Date the project record was created.	**x**	**x**

		(**Security** block)							
createdOnBehalfOf	**CREATED_ON_BEHALF_OF_ID**			**x**	TContact	object	When not null, it indicates that the record was created through an e-billing system or other external application. The contact referenced will often be a vendor in TeamConnect.	**X**	
currentPhaseType-->	**CURRENT_PHASE_TYPE_ID** (**NUMBER**)	**Current Phase** (**General** block)	**x**	**x**	WObjdPhaseType	object	8 The current phase of the project. --> Links to the definition of the phase.	**x**	
defaultCategory-->	**DEFAULT_CATEGO RY_ID** (**NUMBER**)	Indicated by a blue diamond (**Categories** block)	**x**	**x**	YRecentlyViewedRecord	object	8 The default category for the project. --> Links to the category's definition information.	**x**	**x**
categories--> Before TeamConnect 3.3 SP2: detailList-->	**n/a**		**x**	**x**	EProjDetail In Object Navigator, links to Category list which appears in the UI only.		8 The list of categories added to the current object. --> In the object model, the added categories link to the values that have been added for custom fields,	**x**	

							according to field type. --> In Object Navigator, this attribute is enhanced. It links to a list of all categories defined for the object definition, rather than those added to the current object. From there, you can select a category and traverse to the list of custom fields that belong to it.		
documentFolder-->	**DOCUMENT_FOLDER_ID** **(NUMBER)**			**x**	TDocument	object	Links to specific information about the document folder.		
expenseList-->	**n/a**	**Expenses** tab	**x**	**x**	TExpense		8 The list of expenses that are related to the project. --> Links to specific expense information. In Object Navigator, links to a table named **Category** that allows you to filter the expenses based on a		

							category, if desired.		
groupAccessList-->	**n/a**			**x**	EProjGroupAccess		Links to the group security information entered through the Security block of the record.	**x**	
historyList-->	**n/a**	**History** tab	**x**	**x**	THistory		8 The list of histories that are related to the project. --> Links to specific history information for all of the project's history records.	**x**	
involvedFolder-->	**INVOLVED_FOLDER_ID** **(NUMBER)**			**x**	TDocument	object	Links to the folder that stores documents that belong to involved parties for the project.		
involvedList-->	**n/a**	**Involved** tab	**x**	**x**	TInvolved		8 The list of involveds that are related to the project. --> Links to specific involved party information for involved parties of the project. In Object Navigator, links to a table named **Involved Role** that allows you	**x**	

							to filter the involved parties based on a role, if desired.		
isClosed	**IS_CLOSED** **(NUMBER)**		**x**			bool ean	The setting for whether a project is in the open or closed status. 1 - is Closed 0 - is Open	**x**	
isElectronic	**IS_ELECTRONIC**		**x**			bool ean	If TRUE, indicates that the record originated from an external application such as Collaborati.		
relations--> Before TeamConnect 3.3 SP2: leftRelationList-->	**n/a**	**Relations** tab	**x**	**x**	ProjRelation		8 The list of relations in which the related project record is on the left side of the relation, and the current project record is on the right side of the relation. --> Links to specific relation information. In Object Navigator, this attribute links to an intermediary table **Relation Type** that allows you to filter the	**x**	

							relations by relation type, if desired. ***Note:*** *The distinction between right and left project in TeamConnect 3.4 is determined by a 3.4-specific attribute named* ***relationDirection****.*		
mainAssignee-->	**MAIN_ASSIGNEE_ID** **(NUMBER)**			**x**	JProjAssignee	object	Links to the information about the main assignee only, such as the main assignee's user account or the assigned on date.	**x**	**x**
modifiedBy-->	**MODIFIED_BY** **(DATE)**	**Modified By** (**Security** block)	**x**	**x**	YUser	object	8 User who last modified the record. --> Links to specific		

							information about the user account.		
modifiedOn	**MODIFIED_ON** **(DATE)**	**Modified On** (**Security** block)	**x**			date	Date the record was last modified.		
name	**NAME** **(VARCHAR2) (250)**	**Name** (**General** block)	**x**			string (250)	The name of the record.		**x**
nameUpper	**NAME_UPPER** **(VARCHAR2) (250)**		**x**			string (250)	The name of the record in uppercase characters.		
idNumber **Before TeamConnect 3.3 SP2: numberString**	**NUMBER_STRING** **(VARCHAR2) (50)**	**Number** (**General** block)	**x**			string (50)	The record number.		**x**
numberStringUpper	**NUMBER_STRING_ UPPER** **(VARCHAR2) (50)**		**x**			string (50)	The record number in uppercase characters.		
openedOn	**OPENED_ON** **(DATE)**	**Opened On** (**General** block)	**x**			date	Date the project was opened.		
parent-->	**PARENT_ID**		**x**	**x**	TProject, if a parent exists	object	8 The parent project of the current project.	**x**	**x**

	(NUMBER)						--> Links to specific information about the project record.		
phaseList--> **Before TeamConnect 3.3 SP2: phaseChanges-->**	**n/a**			**x**	JProjPhase		Links to the table that contains information for all phases for the project that have taken place.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the project record.		
relations--> **Before TeamConnect 3.3 SP2: rightRelationList-->**	**n/a**	**Relations** tab	**x**	**x**	ProjRelation		8 The list of relations in which the related project record is on the right side of the relation, and the current project record is on the left side of the relation. --> Links to specific relation information. In Object Navigator, this attribute links to an intermediary table **Relation Type** that allows you to filter the relations by	**x**	

							relation type, if desired. ***Note:*** *The distinction between right and left project in Team Connect 3.4 is determined by a 3.4-specific attribute named* ***relationDirection****.*		
securityTypeIID	**SECURITY_TYPE_IID** **(NUMBER)**	**Public/Private** (**Security** block)	**x**			Enum	Specifies whether the project is public or private as follows: 0 - Public (PUBLIC) 2 - Private (PRIVATE)		
taskList-->	**n/a**	**Tasks** tab	**x**	**x**	TTask		8 The list of tasks that are related to the project. --> Links to specific task	**x**	

							information for tasks that are related to the project. In Object Navigator, links to a table named **Category** that allows you to filter the tasks based on a category, if desired.		
treeKey	**TREE_KEY** **(VARCHAR2) (700)**		**x**			string (700)	The chain of primary keys that identifies the parent tree of the project. Used for searching and reporting purposes only.		
userAccessList-->	**n/a**			**x**	EProjUserAccess		Links to the user security information entered through the Security block of the record.	**x**	
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the project record has been updated.		

1.1.27.2 JProjAssignee

JProjAssignee contains information about the assignees in a project record, such as the assigned user, date assigned, project to which the assignee belongs, and assignee's role (such as adjuster). This table represents the **Assignees** block on the **Assignees** tab of a project record.

Object: JProjAssignee (J_PROJ_ASSIGNEE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
assignedOn	**ASSIGNED_ON** **(DATE)**	**Assigned On** (**Assignees** block)	**x**			date	The date the user was assigned to the project.		
isActive	**IS_ACTIVE** **(NUMBER)**	**Status** (**Assignees** block)	**x**			boolean	Indicates whether the assignee is active. 1 - is Active 0 - not Active		**x**
owner-->	**OWNER_ID** **(NUMBER)**		**x**	**x**	TProject	object	8 The project to which the assignee belongs. --> Links to specific information about the project record.		**x**
primary Key	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the assignee.		
relationType--> Before TeamConnect 3.3 SP2: type-->	**TYPE_ID** **(NUMBER)**	**Role** (**Assignees** block)	**x**	**x**	LProjAssigneeType	object	8 The type (or role) of the assignee. --> Links to the available assignee types.		**x**
unassignedOn	**UNASSIGNED_O**	**Unassigned On**	**x**			date	The date the user was un-		

	N **(DATE)**	(**Assignees** block)					assigned.		
user-->	**USER_ID** **(NUMBER)**	**Assignee** (**Assignees** block)	**x**	**x**	YUser	object	8 The user who is an assignee. --> Links to specific information about the user account.		**x**
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the assignee has been updated.		

1.1.27.2.1 LProjAssigneeType

LProjAssigneeType contains information that defines project assignee roles. This information includes the name of the role, tree position, and custom object definition to which it belongs. This table represents the **Assignee Role Type** system lookup table, which appears on the **Assignee Roles** tab of custom object definitions.

Object: LProjAssigneeType (L_PROJ_ASSIGNEE_TYPE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
embeddedProjects--> BeforeTeamConnect 3.3 SP2: childList-->	**n/a**		**x**	**x**	LProjAssigneeType		8 The list of all assignee roles that are child roles to the currently selected role. --> Links to specific information about assignee roles that are		

							defined for the project object.		
displayOrder	**DISPLAY_ORDER** **(NUMBER)**	Order (Assignee Roles tab)	**x**			int	Indicates the order of this assignee role in the list of assignee roles in the GUI.		
name	**NAME** **(VARCHAR2) (50)**	Role Name (Assignee Roles tab)	**x**			string (50)	The name of the assignee role.		
parent-->	**PARENT_ID** **(NAME)**		**x**	**x**	LProjAssigneeType	object	8 The parent assignee role of the currently selected role. --> Links to specific information about assignee roles that are defined for the project object.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the assignee role.		
treePosition	**TREE_POSITION** **(VARCHAR2) (250)**	Tree Position (Assignee Roles tab)	**x**			string (250)	Indicates the tree position for the assignee role.		
isActive	**IS_ACTIVE** **(NUMBER)**	**Status** (**Assignees** block)	**x**			boolean	Indicates whether the assignee is active.		**x**

							1 - is Active 0 - not Active		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the assignee role has been updated.		

1.1.27.3 JProjPhase

JProjPhase contains information about the phases that have occurred in a project record. This includes the phase type (such as open or close), duration, transition date, and user who made the transition. This table represents the Phase History block on the Phases tab of a project record.

Object: JProjPhase (J_PROJ_PHASE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
owner-->	**PROJECT_ID** **(NUMBER)**		**x**	**x**	TProject	object	8 The project for which the phase has been selected. --> Links to specific information about the project record.		
phaseDuration	**PHASE_DURATION** **(NUMBER)**	**Duration** (**Phases** block in project record)	**x**			int	The duration of the phase.		

primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the phase.		
transitionedBy-->	**TRANSITIONED_BY_ID** **(NUMBER)**	**Set By** (**Phases** block in project record)	**x**	**x**	YUser	object	8 The user who changed the project to this phase. --> Links to specific information about the user account.		
transitionedOn	**TRANSITIONED_ON** **(DATE)**	**Set On** (**Phases** block in project record)	**x**			date	The date on which the phase was selected for the project (or, the date that the phase was changed).		
relationType--> Before TeamConnect 3.3 SP2: type-->	**TYPE_ID** **(NUMBER)**	**Phase** (**Phases** block in project record)	**x**	**x**	WObjdPhaseType	object	8 The phase type that was selected for the record. --> Links to specific information about phases defined for the project object.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates the number of times the phase has been updated.		

1.1.27.3.1 WObjdPhaseType

WObjdPhaseType contains the information that defines project phase types. This includes the name of the phase and the unique code. This table also links to lists of phase transitions for which the phase type is the beginning phase or resulting phase. This table represents the Phases block on the Phases tab of custom object definitions.

Object: WObjdPhaseType (W_OBJD_PHASE_TYPE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
displayOrder	**DISPLAY_ORDER** (NUMBER)	**Order** (**Phases** tab in Object Definition)	**x**			int	The order in which the phase appears in the list of phases in the **Phase Transitions** tab.		
fromPhaseTransitionList-->	**n/a**		**x**	**x**	WObjdPhaseTransition		8 The list of all phase transitions in which this phase is the phase from which the transition begins. --> Links to the specific information for phase transitions in which this phase is the phase from which the transition takes place.		
name	**NAME** (VARCHAR2) (250)	**Phase Name** (**Phases** tab in	**x**			string (250)	The name of the phase.		

		Object Definition)							
primaryKey	**PRIMARY_KEY** (NUMBER)		**x**			int	The unique ID of the phase.		
toPhaseTransitionList-->	**n/a**		**x**	**x**	WObjdPhaseTransition		8 The list of all phase transitions in which this phase is the phase to which the transition goes. --> Links to the specific information for phase transitions in which this phase is the phase to which the transition goes.		
uniqueCode	**UNIQUE_CODE** (VARCHAR2) (4)	**Unique Code** (**Phases** tab in Object Definition)	**x**			string (4)	The unique four-character alphanumeric code that is created when the phase is defined.		
version	**VERSION** (NUMBER)		**x**			int	Indicates the number of times the phase has been updated.		

1.1.27.3.2 WObjdPhaseTransition

WObjdPhaseTransition contains information about phase transitions, such as the name and the starting and ending phase. This table represents the Phase Transitions block on the Phase Transitions tab of custom object definitions.

Object: WObjdPhaseTransition (W_OBJD_PHASE_TRANSITION)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
createdBy-->	**CREATED_BY_ID** (NUMBER)		x	x	YUser	object	8 The user who created the phase transition. --> Links to specific information about the user account.		
createdOn	**CREATED_ON** (DATE)		x			date	The date the phase transition was defined.		
leftProjectPhase-->	**LEFT_PROJECT_PHASE_ID** (NUMBER)	**From Phase** (**Phase Transitions** tab)	x	x	WObjdPhaseType	object	8 The phase from which the phase transition begins. --> Links to specific information about the phase definition.		
modifiedBy-->	**MODIFIED_BY_ID** (NUMBER)		x	x	YUser	object	8 The user who last modified the phase transition. --> Links to specific information about the user account.		

modifiedOn	**MODIFIED_ON** (DATE)		x			date	The date the phase transition was last modified.		
name	**NAME** (VARCHAR2) (250)	**Name**	x			string (250)	The name of the phase transition.		
primaryKey	**PRIMARY_KEY** (NUMBER)		x			int	The unique ID of the phase transition.		
rightProjectPhase-->	**RIGHT_PROJECT_PHASE_ID** (NUMBER)	**To Phase** (**Phase Transitions** tab)	x	x	WObjdPhaseType	object	8 The phase at which the phase transition ends. --> Links to specific information about the phase definition.		
version	**VERSION** (NUMBER)		x			int	Indicates the number of times the phase transition has been updated.		

1.1.27.4 JProjRelation

JProjRelation contains information about the project relations in a project record.

This information includes the project on the left and right sides of the relationship and the type of relationship between the two projects (such as related). This table represents the Relations block on the Relations tab of a project record.

Object: ProjRelation (J_PROJ_RELATION)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz

leftProject-->	**LEFT_PROJECT_ID** **(NUMBER)**	Project (left side, **Relations** block)		**x**	TProject	object	The project on the left side of the relationship.		**x**
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of this item.		
rightProject-->	**RIGHT_PROJECT_ID** **(NUMBER)**	Project (right side, **Relations** block)		**x**	TProject	object	The project on the right side of the relationship.		**x**
relationType--> Before TeamConnect 3.3 SP2: type-->	**TYPE_ID** **(NUMBER)**	Relation (**Relations** block)	**x**	**x**	LProjRelationType	object	8 The type of relationship between the two projects. --> Links to specific definition information for the relation.		**x**
version	**VERSION** **(NUMBER)**		**x**			int	Indicates the number of times the item has been updated.		

1.1.27.4.1 LProjRelationType

LProjRelationType contains information about the project relation types defined for custom objects, such as the name of the relation and the tree position. This table represents the **Project Relation Type** system lookup table.

Object: LProjRelationType (L_PROJ_RELATION_TYPE)

Attribute	Database column name	Field in UI	End of	Bridge	Links to	Data	Comments	Commonly used in:

			path		object table:	type		Rules	
childList-->	**n/a**						Do not apply. These lookup items cannot have a hierarchical structure.		
parent-->	**PARENT_ID** (**NUMBER**)								
displayOrder	**DISPLAY_ORDER** (**NUMBER**)	**Order** (**System Lookup Tables**, **Project Relation Type**)	**x**			int	Indicates the order in which this relation appears in the **Type of relationship** list in the **Relations** block of a project.		
name	**NAME** (**VARCHAR2**) **(50)**	**Item Name** (**System Lookup Tables**, **Project Relation Type**)	**x**			string (50)	The name of the relationship type.		
primaryKey	**PRIMARY_KEY** (**NUMBER**)		**x**			int	The unique ID of this item.		
treePosition	**TREE_POSITION** (**VARCHAR2**) **(250)**	**Tree Position** (**System Lookup Tables**, **Project Relation Type**)	**x**			string (250)	The unique four-character alphanumeric code that is created when the relation is defined.		
version	**VERSION**		**x**			int	Indicates the number of times		

	(NUMBER)						the item has been updated.		

1.1.27.5 EProjDetail

EProjDetail contains all categories added to project records. Each time a category is added to a project record, an entry is made in this table in the database.

In Object Navigator, a list of all categories that have been defined for the current object definition appears. This list is labeled Category. From each category, you can traverse to a table called Detail Fields, which displays all custom fields belonging to the category you selected.

For more information about how custom field values are stored in the database, see Object Model: Custom Fields.

Object: EProjDetail (E_PROJ_DETAIL)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
category-->	**CATEGORY_ID** (NUMBER)	**Category**	**x**	**x**	YRecentlyViewedRecord	object	8 The category definition to which this category, which has been added to a record, belongs. --> Links to specific information about the category.		
isManual	**IS_MANUAL** (NUMBER)		**x**			boolean	Specifies whether the category is added manually or automatically to the project. Categories are added automatically when they are the parent of the category that is added manually.		

							0 - Added Automatically 1 - Added Manually		
owner -->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	TProject	object	8 The project to which the category belongs. --> Links to the information about the project.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the project category.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the category was updated.		

1.1.27.6 EProjUserAccess

EProjUserAccess contains the user access information that is set in the Security block of each project record. For example, it includes whether each user that is listed in the Security block is allowed or denied the Read, Update, Delete, and Perm(ission) rights.

Object: EProjUserAccess (E_PROJ_USER_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ** **(NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected).		

							This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM (NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		

allowDenyIID	**ALLOW_DENY_IID** **(CHAR) (1)**	**Option**	**x**			Enum	Indicates whether the user is allowed or denied access to the record. a - Allow (ALLOW) d - Deny (DENY)		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the user rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	TProject	object	8 The project to which this user access setting belongs. --> Links to specific information about the project record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the user access rights.		
user-->	**USER** **(VARCHAR2) (250)**	**List**	**x**	**x**	YUser	object	8 The user the access rights pertain to. --> Links to specific information about the user account.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the user access rights have been updated.		

1.1.27.7 EProjGroupAccess

EProjGroupAccess contains the group access information that is set in the **Security** block of each project record. For example, it includes whether each group that is listed in the **Group** field of the **Group Rights** block on the **Security** tab is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EProjGroupAccess (E_PROJ_GROUP_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ** **(NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE** **(NUMBER)**	Update check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE** **(NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed		

							or denied. (See allowDenyIID.)		
isPerm	**IS_PERM** **(NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(CHAR) (1)**		**x**			Enum	Indicates whether the group is allowed or denied access to the record. a - Allow (ALLOW) d - Deny (DENY)		
group-->	**GROUP_ID** **(NUMBER)**		**x**	**x**	YGroup	object	8 The group that the access rights pertain to. --> Links to specific information about the group account.		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the group rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID**		**x**	**x**	TProject	object	8 The project to which this group access setting belongs.		

	(NUMBER)						--> Links to specific information about the project record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of this item.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates the number of times the item has been updated.		

1.1.27.8 TInvolved

TInvolved contains the general information of all involved party records. This these records include the contact who is the involved party, project the involved record belongs to, default role, and users who created and modified each record.

Object: TInvolved (T_INVOLVED)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	**Temp/Wiz**
activeOn	**ACTIVE_ON** **(DATE)**		**x**			date	The date on which the involved party became active.		
application-->	**APPLICATION_ID** **(NUMBER)**		**x**	**x**	WObjdProjectInfo	int	8 The Custom Object definition to which this involved belongs. --> Links to the table that contains information that		

							is specific to project object definitions (such as phase definitions, object autonumbering, and object naming pattern) and indirectly relates the involved to the involved object definition.		
contact-->	**CONTACT_ID** (**NUMBER**)	**Contact** (**General** block)	**x**	**x**	TContact	object	8 The contact who is selected as the involved party for this involved record. --> Links to specific information about the contact record.	**x**	**x**
createdBy-->	**CREATED_BY_ID** (**NUMBER**)	**Created By** (**Security** block)	**x**	**x**	YUser	object	8 User who created the involved record. --> Links to specific information about the user account.	**x**	**x**
createdOn	**CREATED_ON** (**DATE**)	**Created On** (**Security** block)	**x**			date	Date the involved record was created.	**x**	**x**
createdOnBehalfOf	**CREATED_ON_BEHALF_OF_ID**			**x**	TContact	object	When not null, it indicates that the record was created through an e-billing system or other	**X**	

							external application. The contact referenced will often be a vendor in TeamConnect.		
defaultCategory-->	**DEFAULT_CATEGORY_ID (NUMBER)**	Indicated by a blue diamond (**Categories** block)	**x**	**x**	YRecentlyViewedRecord	object	8 The default role for the involved. --> Links to specific information about the role. Note: Involved roles are the equivalent of categories for other main objects. This is why their definition information is contained in WObjdCategory.	**x**	
categories--> **Before TeamConnect 3.3 SP2: detailList-->**	**n/a**		**x**	**x**	EInvlDetail In Object Navigator, links to Category list which appears in the UI only.		8 The list of categories added to the current object. --> In the object model, the added categories link to the values that have been added for custom fields, according to field type. --> In Object Navigator, this attribute is enhanced. It links to a list of	**x**	**x**

							all categories defined for the object definition, rather than those added to the current object. From there, you can select a category and traverse to the list of custom fields that belong to it.		
documentFolder-->	**DOCUMENT_FOLDER_ID** **(NUMBER)**			**x**	TDocument	object	Links to specific information about the document folder.		
groupAccessList-->	**n/a**			**x**	EInvGroupAccess		Links to the group security information entered through the Security block of the record.		
historyList-->	**n/a**	**History** tab	**x**	**x**	THistory		8 The list of histories that are related to the involved record. --> Links to specific history information for all of the involved's history records.	**x**	
inactiveOn	**INACTIVE_ON** **(DATE)**		**x**			date	The date on which the involved party became inactive.		

isActive	**IS_ACTIVE** **(DATE)**		**x**			int	1 = Involved party is active. 0 = Involved party is inactive.		
relations--> **Before TeamConnect 3.3 SP2: leftRelationList-->**	**n/a**	**Relations** tab	**x**	**x**	InvlRelationAttrMap		8 The list of relations in which the related involved record is on the left side of the relation, and the current involved record is on the right side of the relation. --> Links to specific relation information. In Object Navigator, this attribute links to an intermediary table **Relation Type** that allows you to filter the relations by relation type, if desired. ***Note:*** *The distinction between right and left involved in TeamConnect 3.4 is determined by a 3.4-*		

							specific attribute named ***relationDirection***.		
modifiedBy-->	**MODIFIED_BY_ID** **(NUMBER)**		**x**	**x**	YUser	object	8 The user who last modified the record. --> Links to specific information about the user account.		
modifiedOn	**MODIFIED_ON** **(DATE)**		**x**			date	The date on which the record was last modified.		
relations--> Before TeamConnect 3.3 SP2: rightRelationList-->	**n/a**	**Relations** tab	**x**	**x**	InvRelationAttrMap		8 The list of relations in which the related involved record is on the right side of the relation, and the current involved record is on the left side of the relation. --> Links to specific relation information. In Object Navigator, this attribute links to an intermediary table **Relation Type** that allows you to		

							filter the relations by relation type, if desired. ***Note:*** *The distinction between right and left involved in TeamConnect 3.4 is determined by a 3.4-specific attribute named* ***relationDirection****.*		
note-->	**NOTE_ID** **(NUMBER)**	**Notes** (**General** block)		**x**	JNote	object	8 The notes entered for the record. --> Links to the table that contains the text entered into the **Notes** field of the record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the involved record.		
project-->	**PROJECT_ID**		**x**	**x**	TProject	object	8 The project to which this	**x**	**x**

	(NUMBER)						involved record belongs. --> Links to specific information for the project record.		
securityTypeIID	**SECURITY_TYPE_ID** **(NUMBER)**	**Public/ Private** (**Security** block)	**x**			En um	Specifies whether the involved record is public or private. 0 - Public (PUBLIC) 2 - Private (PRIVATE)		
userAccessList-->	**n/a**			**x**	EInvlUserAccess		Links to the user security information entered through the Security block of the record.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the involved record has been updated.		

1.1.27.8.1 JInvlRelation

JInvlRelation contains the information of all relations between involved parties, such as the type of relationship and the involved contacts on the left and right sides of the relation.

Object: InvlRelationAttrMap (J_INVL_RELATION)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/

									Wiz
leftInvolved-->	**LEFT_INVOLVED_ID (NUMBER)**	**Involved Contact** (left side, **Relations** block)		**x**	TInvolved	object	The involved that is on the left side of the relationship.		**x**
rightInvolved-->	**RIGHT_INVOLVED_ID (NUMBER)**	**Involved Contact** (right side, **Relations** block)		**x**	TInvolved	object	The involved that is on the right side of the relationship.		**x**
relationType--> BeforeTeamConnect 3. 3 SP2: type-->	**TYPE_ID (NUMBER)**	**Relationship** (**Relations** block)	**x**	**x**	LContRelationType	object	8 The type of the relationship. --> Links to definition information for the relation type.		**x**
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the involved relation item was updated.		

1.1.27.8.2 EInvlDetail

EInvlDetail contains all categories added to involved records. Each time a category is added to an involved record, an entry is made in this table in the database.

In Object Navigator, a list of all categories that have been defined for the current object definition appears. This list is labeled **Category**. From each category, you can traverse to a table called **Detail Fields**, which displays all custom fields belonging to the category you selected.

For more information about how custom field values are stored in the database, see Object Model: Custom Fields.

Object: EInvlDetail (E_INVL_DETAIL)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp / Wiz
category-->	**CATEGORY_ID** **(NUMBER)**	**Category**	**x**	**x**	YRecentlyViewedRecord	object	8 The category definition to which this category, which has been added to a record, belongs. --> Links to specific information about the category.		
owner -->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**			**x**	TInvolved	object	Links to the involved record to which the category has been added.		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Specifies whether the category is added manually or automatically to the involved. Categories are added automatically when they are the parent of the category that is added manually. 0 - Added Automatically 1 - Added Manually		

primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the involved category record.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the added category was updated.		

1.1.27.8.3 EInvlUserAccess

EInvlUserAccess contains the user access information that is set in the **Security** block of each involved record. For example, it includes whether each user that is listed in the **Security** block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EInvlUserAccess (E_INVL_USER_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected).		

							This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE** **(NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM** **(NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(CHAR) (1)**	**Option**	**x**			Enum	Indicates whether the user is allowed or denied access to the record. a - Allow (ALLOW) d - Deny (DENY)		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the user rights were assigned manually		

							through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	TInvolved	object	8 The involved record to which this user access setting belongs. --> Links to specific information about the involved record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the user access rights.		
user-->	**USER_ID** **(NUMBER)**	**List**	**x**	**x**	YUser	object	8 The user the access rights pertain to. --> Links to specific information about the user account.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the user access rights have been updated.		

1.1.27.8.4 EInvlGroupAccess

EInvlGroupAccess contains the group access information that is set in the **Security** block of each involved record. For example, it includes whether each group that is listed in the **Security** block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EInvlGroupAccess (E_INVL_GROUP_ACCESS)

Attribute	Database column name	Field in UI	End of	Bridge	Links to obje	Data type	Comments	Commonly used in:

			path		ct table:			Rules	
isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are		

							allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM** **(NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(CHAR) (1)**	**Option**	**x**			Enum	Indicates whether the group is allowed or denied access to the record. a - Allow (ALLOW) d - Deny (DENY)		
group-->	**GROUP_ID** **(NUMBER)**	**List**	**x**	**x**	YGroup	object	8 The group that the access rights pertain to. --> Links to specific information about the group account.		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the group access rights were assigned manually through the Security block of the record or automatically by the system.		

							0 - Assigned manually 1 - Assigned automatically		
owner--->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	TInvolved	object	8 The involved record to which this group access setting belongs. --> Links to specific information about the involved record.		
primary Key	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the group access rights.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the group access rights have been updated.		

1.1.27.9 TMilestone

TMilestone contains the general information for all milestone records.

This information includes the projected, extension, and completion dates, milestone description, the project the milestone belongs to, as well as the users who created and modified the milestone record. This table also links to the list of custom fields created for the milestone categories and other related information contained in other tables.

Object: TMilestone (T_MILESTONE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
application-->	**APPLICATION_ID**		**x**	**x**	WObjdProjectInfo	object	8 The Custom Object		

	(NUMBER)						definition to which this milestone belongs. --> Links to the table that contains information that is not included in ZObjectDefinition because it is only necessary for Custom Object Definitions (such as phase definitions, object autonumbering, and object naming pattern).		
completionDate	**COMPLETION_DATE** **(DATE)**	Completion Date (General block)	**x**			date	The date the milestone was completed.		**x**
completionEventID	**COMPLETION_EVENT_ID** **(NUMBER)**	phase is Started/ Completed (General block)	**x**			Enum	Indicates whether the milestone should be automatically marked as completed when the related project phase (selected in the phase--> attribute) is		

							started or completed. 0 - Started (STARTED) 1 - Completed (COMPLETED)		
createdBy-->	**CREATED_BY_ID** **(NUMBER)**	Created By (Security block)	**x**	**x**	YUser	object	8 User who created the milestone record. --> Links to specific information about the user account.	**x**	**x**
createdOn	**CREATED_ON** **(DATE)**	Created On (Security block)	**x**			date	Date the milestone record was created.	**x**	**x**
createdOnBehalfOf	**CREATED_ON_BEHALF_OF_ID**			**x**	TContact	object	When not null, it indicates that the record was created through an e-billing system or other external application. The contact referenced will often be a vendor in TeamConnect.	**X**	

defaultCategory-->	**DEFAULT_CATEGORY_ID** **(NUMBER)**	Indicated by a blue diamond (Categories block)	**x**	**x**	YRecentlyViewedRecord	object	8 The default category for the milestone. --> Links to specific information about the category.		**x**
detailList-->	**n/a**		**x**	**x**	EMileDetail In Object Navigator, links to Category list which is displayed in the UI only.		8 The list of categories added to the current object. --> In the object model, the added categories link to the values that have been added for custom fields, according to field type. --> In Object Navigator, this attribute is enhanced. It links to a list of all categories defined for the object definition, rather than those added to the current object. From there, you can select a category and traverse to the list of custom fields		

							that belong to it.		
documentFolder-->	**DOCUMENT_FOLDER_ID (NUMBER)**			**x**	TDocument	object	Links to specific information about the document folder.		
extensionDate	**EXTENSION_DATE (DATE)**	Extension (General block)	**x**			date	The extended date of the milestone.	**x**	
groupAccessList-->	**n/a**			**x**	EInvGroupAccess		Links to the group security information entered through the Security block of the record.		
historyList-->	**n/a**	History tab	**x**	**x**	THistory		8 The list of histories that are related to the milestone record. --> Links to specific history information for all of the milestone's history records.		
isPhaseUsed	**IS_PHASE_USED (NUMBER)**	Mark this milestone complete	**x**			boolean	Indicates whether the milestone record is set to be marked as complete		

		when (check box - General block)					automatically when the related project record reaches a certain phase. 0 - Do not use phase to mark milestone as complete 1 - Use phase to mark milestone as complete		
milestone Date	**MILESTONE_DATE** **(DATE)**		**x**			date	This contains one of the last of the three dates for the milestone record in the following order. Projected Date Extended Date Completion Date For example, if the Projected Date and Extended Date have values, then the Milestone Date will be populated with the		

							same date as the Extended Date.		
modifiedBy-->	**MODIFIED_BY_ID** **(NUMBER)**		**x**	**x**	YUser	object	8 The user who last modified the record. --> Links to specific information about the user account.		
modifiedOn	**MODIFIED_ON** **(DATE)**		**x**			date	The date on which the record was last modified.		
note-->	**NOTE_ID** **(NUMBER)**			**x**	JNote	object	Links to the table that contains the text entered into the **Notes** field of the record.		
phase-->	**PHASE_ID** **(NUMBER)**		**x**	**x**	WObjdPhaseType	object	8 The phase of the related project record at which the milestone will be automatically marked as complete. --> Links to definition information for the phase.	**x**	
primaryKey	**PRIMARY_KEY**		**x**			int	The unique ID of the		

	(NUMBER)						milestone record.		
project-->	**PROJECT_ID** **(NUMBER)**		**x**	**x**	TProject	object	8 The project to which this milestone belongs. --> Links to specific information about the project record.		**x**
projectedDate	**PROJECTED_DATE** **(DATE)**	Projected Date (General block)	**x**			date	The projected date of completion of the milestone.	**x**	**x**
securityTypeIID	**SECURITY_TYPE_I ID** **(NUMBER)**	Public / Private (Security block)	**x**			Enum	Specifies whether the milestone record is public or private. 0 - Public (PUBLIC) 2 - Private (PRIVATE)		
shortDescription	**SHORT_DESCRIPT ION** **(VARCHAR2) (250)**	Subject (General block)	**x**			string (250)	The subject (or description) of the milestone record.		**x**
userAccessList-->	**n/a**			**x**	EInvUserAccess		Links to the user security information entered through the Security		

							block of the record.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the milestone record has been updated.		

1.1.27.10 EMileDetail

EMileDetail contains all categories added to milestone records. Each time a category is added to a milestone record, an entry is made in this table in the database.

In Object Navigator, EMileDetail is not displayed when you traverse using the **detailList-->** bridge. Instead, a list of all categories that have been defined for the object definition is displayed. This list is labeled **Category**. From each category, you can traverse to a table called **Detail Fields**, which displays all custom fields belonging to the category you selected.

For more information about how custom field values are stored in the database, see Object Model: Custom Fields.

Object: EMileDetail (E_MILE_DETAIL)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
category-->	**Category_ID (NUMBER)**	**Category**	**x**	**x**	YRecentlyViewedRecord	object	8 The category of the milestone with which the details are associated. --> Links to specific information about the category.		
owner-->	**ENTERPRISE_OBJECT_ID (NUMBER)**			**x**	TMilestone	object	The milestone with which the custom fields are associated.		

							Links to the information about the milestone record.		
isManual	**IS_MANUAL (NUMBER)**		**x**			boolean	Specifies whether the category is added manually or automatically to the milestone. Categories are added automatically when they are the parent of the category that is added manually. 0 - Added Automatically 1 - Added Manually		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the milestone category record.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the added category was updated.		

1.1.27.11 EMileUserAccess

EMileUserAccess contains the user access information that is set in the **Security** block of each milestone record.

For example, it includes whether each user that is listed in the **Security** block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EMileUserAccess (E_MILE_USER_ACCESS)

Attribute	Database column	Field in	End	Bridg	Links to	Data type	Comments	Commonly used in:

	name	UI	of path	e	object table:			Rules	
isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		

isPerm	**IS_PERM** **(NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(CHAR) (1)**	**Option**	**x**			Enum	Indicates whether the user is allowed or denied access to the record. a - Allow (ALLOW) d - Deny (DENY)		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the user rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	TMilestone	object	8 The milestone record to which this user access setting belongs. --> Links to specific information about the milestone.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the user access rights.		

user-->	**USER_ID (NUMBER)**	**List**	**x**	**x**	YUser	object	8 The user the access rights pertain to. --> Links to specific information about the user account.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the user access rights have been updated.		

1.1.27.12 EMileGroupAccess

EMileGroupAccess contains the group access information that is set in the Security block of each milestone record. For example, it includes whether each group that is listed in the Security block is allowed or denied the Read, Update, Delete, and Perm(ission) rights.

Object: EMileGroupAccess (E_MILE_GROUP_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1		

							(selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE** **(NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM** **(NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(CHAR) (1)**	**Option**	**x**			Enum	Indicates whether the group is allowed or denied access to the record. a - Allow (ALLOW) d - Deny (DENY)		

group-->	**GROUP_ID (NUMBER)**	**List**	**x**	x	YGroup	object	8 The group that the access rights pertain to. --> Links to specific information about the group account.		
isManual	**IS_MANUAL (NUMBER)**		**x**			boolean	Indicates whether the group access rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID (NUMBER)**		**x**	x	TMilestone	object	8 The milestone record with which this group access setting is associated. --> Links to specific information about the milestone record.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	Unique ID for the group access rights.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the group access rights have been updated.		

1.1.28 Object Model: Contacts

The tables in this appendix provide information about the object model as it relates to contacts.

Note: *Designer contact groups are referred to as Address Books in the end-user interface.*

1.1.28.1 TContact

TContact contains general information for contact records. This information includes the contact's name, birth date, default category, driver license number, title, Social Security number or Tax ID, the company the person works for, whether the contact is a person or a company, and so on. It also links to the lists of addresses, fax and phone numbers, rates, and skills.

However, the default address, default phone, default email, and so on are identified by specific attributes of TContact, rather than through the listing attributes (for example, **addressList-->** and **faxList-->**).

Object: TContact (T_CONTACT)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
addresses--> **Before TeamConnect 3.3 SP2: addressList-->**	**n/a**		**x**	**x**	JContAddress	object	8 The list of all addresses that are added to the contact. --> Links to the join table that contains information for all addresses for the contact. Select this attribute when you want to access items belonging to	**x**	

							an address, such as city or state, or when you need to access address types. In Object Navigator, this attribute links to a table named **Address Type** that allows you to filter the addresses based on a type, if desired.		
alias	**ALIAS** **(VARCHAR2)(250)**	**Alias / Nickname** (**General** block)	**x**			string	The contact's nickname (for individual contact and company contact).	**x**	**x**
birthdate **Before TeamConnect 3.3 SP2: birthDate**	**BIRTH_DATE** **(DATE)**	**Date of Birth** (**General** block)	**x**			date	The birth date of the contact. Time-zone-independent.	**x**	
company-->	**COMPANY_ID** **(NUMBER)**	**Company** (**General** block)	**x**	**x**	TContact	int	8 The company associated with the contact, if the contact is a person.	**x**	**x**

							--> Links to specific information about the company contact record.		
createdBy-->	**CREATED_BY_ID (NUMBER)**	**Created By (Security block)**	**x**	**x**	YUser	object	8 The user who created the contact record. --> Links to specific information for the user account.	**x**	**x**
createdOn	**CREATED_ON (DATE)**	**Created On (Security block)**	**x**			date	Date the contact record was created.	**x**	**x**
createdOnBehalfOf	**CREATED_ON_BEHALF_OF_ID**			**x**	TContact	object	When not null, it indicates that the record was created through an e-billing system or other external application. The contact referenced will often be a vendor in TeamConnect.	**X**	
currency-->	**CURRENCY_ID**	**Currency**		**x**	YCurrencyInfo	object	8 The selected currency for		

	(NUMBER)	(**Rates** block)					the contact, which applies to the contact's rates. --> Links to the definition information of the currency.		
primaryAddress--> Before TeamConnect 3.3 SP2: defaultAddress-->	**DEFAULT_ADDRESS_ID (NUMBER)**	Indicated by a blue diamond	**x**	**x**	JContAddress	object	8 The selected default address for the contact. --> Links to specific information entered for the default address.	**x**	**x**
defaultCategory--->	**DEFAULT_CATEGORY_ID (NUMBER)**	Indicated by a blue diamond (Categories block)	**x**	**x**	YRecentlyViewedRecord	object	8 The selected default category for the contact. --> Links to specific information for the default category.	**x**	**x**
primaryEmailAddress--> Before TeamConnect 3.3 SP2: defaultEmail-->	**DEFAULT_EMAIL_ID (NUMBER)**	Indicated by a blue diamond	**x**	**x**	JContEmail	object	8 The selected default email address for the contact. --> Links to specific information entered for the default email address.	**x**	**x**

primaryFaxNumber--> Before TeamConnect 3.3 SP2: defaultFax-->	**DEFAULT_FAX_ID (NUMBER)**	Indicated by a blue diamond	**x**	**x**	JContFax	object	8 The selected default fax number for the contact. --> Links to specific information entered for the default fax number.	**x**	**x**
primaryInternetAddress--> Before TeamConnect 3.3 SP2: defaultInetAddress-->	**DEFAULT_ADDRESS _ID (NUMBER)**	Indicated by a blue diamond	**x**	**x**	JContInetAddress	object	8 The selected default internet address for the contact. --> Links to specific information entered for the default internet address.	**x**	**x**
primaryPhoneNumber--> Before TeamConnect 3.3 SP2: defaultPhone-->	**DEFAULT_PHONE_I D (NUMBER)**	Indicated by a blue diamond	**x**	**x**	JContPhone	object	8 The selected default phone number for the contact. --> Links to specific information entered for the default phone number.	**x**	**x**
defaultRate--> Before TeamConnect 3.3 SP2: defaultRateValue	**DEFAULT_RATE_V A LUE (NUMBER)**	Default Rate Value (Rates	**x**			decimal	The default rate value for the contact.	**x**	**x**

		block)							
detailList-->	**n/a**		**x**	**x**	EContDetail In Object Navigator, links to Category list which appears in the UI only.		8 The list of categories added to the current object. --> In the object model, the added categories link to the values that have been added for custom fields, according to field type. --> In Object Navigator, this attribute is enhanced. It links to a list of all categories defined for the object definition, rather than those added to the current object. From there, you can select a category and traverse to the list of custom fields that belong to it.	**x**	**x**
distributionList-->	**n/a**			**x**	JDistMember		Links to the list of all members in a contact group		

							(Address Book). From there, you can access the information about a contact group, such as whether it is public or private.		
documentFolder-->	**DOCUMENT_FOLDER_ID (NUMBER)**	Documents block		**x**	TDocument	object	Links to the document folder that belongs to the contact.		
driverLicense	**DRIVER_LICENSE (VARCHAR2) (50)**	State Driver's License (General block)	**x**			string (50)	The contact's driver license number.	**x**	
emailAddresses--> Before TeamConnect 3.3 SP2: emailList-->	**n/a**		**x**	**x**	JContEmail		8 The list of all email addresses that are added to the contact. --> Links to the join table that contains information for all email addresses of the contact. Select this attribute when you want to access email	**x**	

							address types. In Object Navigator, this attribute links to a table named **Email Type** that allows you to filter the email addresses based on a type, if desired.		
faxNumbers--> **Before TeamConnect 3.3 SP2: faxList-->**	**n/a**		**x**	**x**	JContFax		8 The list of all fax numbers that are added to the contact. --> Links to the join table that contains information for all fax numbers of the contact. Select this attribute when you want to access fax number types. In Object Navigator, this attribute links to a table named **Fax Type** that allows you to filter the fax numbers based on a type, if desired.	**x**	

firstName	**FIRST_NAME** **(VARCHAR2) (50)**	First (General block)	**x**			string (50)	The first name of the contact.		
firstNameUpper	**FIRST_NAME_UPPER** **(VARCHAR2) (50)**		**x**			string (50)	The contact's first name, in uppercase letters.		
groupAccessList-->	**n/a**			**x**	EContGroupAccess		Links to the group security information entered through the Security block of the record.		
historyList-->	**n/a**	History tab	**x**	**x**	THistory		8 The list of all histories that are related to the contact record. --> Links to specific history information for all of the contact's history records.	**x**	
internetAddresses--> **Before TeamConnect 3.3 SP2: inetAddressList-->**	**n/a**		**x**	**x**	JContInetAddress		8 The list of all internet addresses that are added to the contact record. --> Links to the join table	**x**	

							that contains information for all internet addresses for the contact. Select when you want to access internet address types. In Object Navigator, this attribute links to a table named **Internet Address Type** that allows you to filter the internet addresses based on a type, if desired.		
relations--> Before TeamConnect 3.3 SP2: leftRelationList--><	**n/a**	Relations tab	**x**	**x**	JContRelation		8 The list of relations in which the related contact record is on the left side of the relation, and the current contact record is on the right side of the relation. --> Links to specific relation information.	**x**	

							In Object Navigator, this attribute links to an intermediary table **Relation Type** that allows you to filter the relations based on the relation type, if desired. ***Note:*** *The distinction between right and left relation in TeamConnect 3.4 is determined by a 3.4-specific attribute named* ***relationDirection****.*		
middleName	**MIDDLE_NAME**	Middle	**x**			string	The middle name of the		

	(VARCHAR2) (50)	(General block)				(50)	contact.		
modifiedBy-->	**MODIFIED_BY** **(NUMBER)**	Modified By (Security block)	**x**	**x**	YUser	object	8 The user who last modified the contact record. --> Links to specific information for the user account.		
modifiedOn	**MODIFIED_ON** **(DATE)**	Modified On (Security block)	**x**			date	Date the contact was last modified.		
lastName **Before TeamConnect 3.3 SP2: name**	**NAME** **(VARCHAR2) (250)**	Last or Company Name (General block)	**x**			string (250)	The last name of the contact, if the contact is a person. The name of the contact, if the contact is a company.		
nameUpper	**NAME_UPPER** **(VARCHAR2) (250)**		**x**			string (250)	The last name or company name of the contact, in uppercase letters.		
note-->	**NOTE_ID** **(NUMBER)**	Notes		**x**	JNote	object	Links to the table that contains the text entered		

		(General block)					into the **Notes** field of the record.		
idNumber **Before TeamConnect 3.3 SP2: numberString**	**NUMBER_STRING** **(VARCHAR2) (50)**	ID (General block)	**x**			string (50)	Alphanumeric text that identifies this contact as per your company's standards.		
numberStringUpper	**NUMBER_STRING_UPPER** **(VARCHAR2) (50)**		**x**			string (50)	The employee number in uppercase characters.		
phoneNumbers--> **Before TeamConnect 3.3 SP2: phoneList-->**	**n/a**		**x**	**x**	JContPhone		8 The list of phone numbers that are added to the contact record. --> Links to the join table that contains information for all phone numbers of the contact. Select this attribute when you want to access phone number types. In Object Navigator, this attribute links to a table named **Phone Type** that allows	**x**	

							you to filter the phone numbers based on a type, if desired.		
prefix	**PREFIX** **(VARCHAR2) (50)**	Prefix (General block)	**x**			string (50)	The prefix of the contact's name (such as Mr. or Mrs.)	**x**	
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the contact record.		
rates--> **Before TeamConnect 3.3 SP2: rateList-->**	**n/a**	Rates tab	**x**	**x**	JContRate		8 The list of rates that are added to the contact record. --> Links to the join table that contains information for all rates of the contact. Select this attribute when you want to access specific items related to contact rates, such as rate amount, effective from date, and effective to date.	**x**	

							In Object Navigator, this attribute links to a table named **Task Category** that allows you to filter the rates based on a task category, if desired.		
relations--> Before TeamConnect 3.3 SP2: rightRelationList-->	**n/a**	Relations tab	**x**	**x**	JContRelation		8 The list of relations in which the related contact record is on the right side of the relation, and the current contact record is on the left side of the relation. --> Links to specific relation information. In Object Navigator, this attribute links to an intermediary table **Relation Type** that allows you to filter the relations based on the relation type, if desired.	**x**	

							Note: *The distinction between right and left relation in TeamConnect 3.4 is determined by a 3.4-specific attribute named relationDirection.*		
salutation*	**SALUTATION** **(VARCHAR2) (50)**	Salutation (General block)	**x**			string (50)	Salutation will **not** appear in TeamConnect; this field remains for users upgrading from 2.x versions of TeamConnect and should be ignored by all 4.x users.		

securityTypeIID	**SECURITY_TYPE_IID** **(NUMBER)**	Security Type (Security block)	**x**			Enum	The security type of the contact record (either Public or Private). 0 - Public (PUBLIC) 2 - Private (PRIVATE)	**x**	
skill--> **Before TeamConnect 3.3 SP2: skillList-->**	**n/a**	Skills tab	**x**	**x**	JContSkill		8 The list of skills that are added to the contact record. --> Links to the join table that contains information for all skills of the contact. Select this attribute when you want to access specific items related to skills, such as skill level and skill type. In Object Navigator, this attribute links to a table named **Skill Type** that allows you to filter the skills based on a type, if desired.	**x**	

socialSecurityNumber **Before TeamConnect 3.3 SP2: ssOrTaxNumberString**	**SS_OR_TAX_NUMBER_STRING** **(VARCHAR2) (50)**	Social Security or Tax ID (General block)	**x**			string (50)	The Social Security number of the contact, if the contact is a person. The Tax ID of the contact, if the contact is a company.	**x**	
suffix	**SUFFIX** **(VARCHAR2) (50)**	Suffix (General block)	**x**			string (50)	The suffix of the contact's name (such as Jr., Sr., or II)		
territories--> **Before TeamConnect 3.3 SP2: territoryList-->**	**n/a**	Territories tab	**x**	**x**	JContTerritory		8 The list of territories that are added to the contact record. --> Links to the join table that contains information for all territories for the contact. Select when you want to access territory types.	**x**	
jobTitle **Before TeamConnect 3.3 SP2: title**	**TITLE** **(VARCHAR2) (250)**	Job Title (General block)	**x**			string (250)	The job title of the contact.	**x**	
typeIID	**TYPE_IID**	New Pers	**x**			Enum	The type of the contact.	**x**	

	(CHAR) (1)	on or New Company (Pop-up Menu)					P - Person (PERSON) C - Company (COMPANY)		
userAccessList-->	**n/a**			**x**	EContUserAccess		Links to the user security information entered through the Security block of the record.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the contact record was updated.		

1.1.28.2 JContAddress

JContAddress contains information about the addresses in a contact record. This information includes the city, street, zip code, county, state, address type (such as Home or Business), and the contact record to which the address belongs. This table represents the **Addresses** block on the **General** tab of a contact record.

Object: JContAddress (J_CONT_ADDRESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
city	**CITY (VARCHAR2) (50)**	City	**x**			string (50)	The city in the contact address.	**x**	**x**

countryItem -->	**COUNTRY_ITEM_ID** **(NUMBER)**	Country	**x**	**x**	LCountryItem	object	8 The country in the contact address. --> Links to definition information for the country item.	**x**	**x**
county	**COUNTY** **(VARCHAR2) (50)**	County	**x**			string (50)	The county of the contact address.	**x**	**x**
currentOn	**CURRENT_ON** **(DATE)**	Current as of	**x**			date	The last known date the address was current. Time-zone-independent.		**x**
isCurrent	**IS_CURRENT** **(NUMBER)**		**x**			boolean	Indicates whether the addess is current. 0 - Not Current 1 - Current	**x**	**x**
owner-->	**CONTACT_ID** **(NUMBER)**		**x**	**x**	TContact	object	8 The contact to which the address belongs. --> Links to specific information about the contact record.		**x**
postalCode	**POSTAL_CODE** **(VARCHAR2) (50)**	Zip	**x**			string (50)	The zip code in the contact address.	**x**	**x**
primaryKey	**PRIMARY_KEY**		**x**			int	The unique ID of the contact		

	(NUMBER)						address.		
referenceNumberString	**REFERENCE_NUMBER_STRING** **(VARCHAR2) (50)**	Guide No.	**x**			string (50)	The reference number for the contact address.		**x**
state	**STATE** **(VARCHAR2) (50)**	State/ Province	**x**			string (50)	The U.S. state in the contact address.	**x**	**x**
street	**STREET** **(VARCHAR20) (250)**	Street	**x**			string (250)	The street address in the contact address.	**x**	**x**
type-->	**TYPE_ID** **(NUMBER)**	Type	**x**	**x**	LContAddressType	object	8 The type of the contact address (such as Home or Business). --> Links to definition information for the address type.	**x**	**x**
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the contact address was updated.		

1.1.28.2.1 LCountryItem

LCountryItem contains information about the country items defined for contact addresses. This includes the name of the country item, tree position, and display order. This table represents the **Country Item** system lookup table.

Object: LCountryItem (L_COUNTRY_ITEM)

Attribute	Database column name	Field in UI	End of pat	Bridge	Links to object	Data typ	Comments	Commonly used in:

								Rules	Temp/Wiz
childList-->	**n/a**						Do not apply. These lookup items cannot have a hierarchical structure.		
parent-->	**PARENT_ID (NUMBER)**								
displayOrder	**DISPLAY_ORDER (NUMBER)**	**Order**	**x**			int	The order in which the table items will be listed in the GUI, lowest number first.		
name	**NAME (VARCHAR2) (50)**	**Item Name**	**x**			string (50)	The name of the country.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the country list item.		
treePosition	**TREE_POSITION (VARCHAR2) (250)**	**Tree Position**	**x**			string (250)	The unique four-character alphanumeric code that is created when the country list item is defined.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the country list item was updated.		

1.1.28.2.2 LContAddressType

LContAddressType contains information about the address types defined for contact addresses. This information includes the name of the address type, tree position, and display order. This table represents the **Address Type** system lookup table.

Object: LContAddressType (L_CONT_ADDRESS_TYPE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
childList-->	**n/a**						Do not apply. These lookup items cannot have a hierarchical structure.		
parent-->	**PARENT_ID (NUMBER)**								
displayOrder	**DISPLAY_ORDER (NUMBER)**	Order	**x**			int	The order in which the table items will be listed in the GUI, lowest number first.		
name	**NAME (VARCHAR2) (50)**	Item Name	**x**			string (50)	The name of the address type.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the address type list item.		
treePosition	**TREE_POSITION (VARCHAR2) (250)**	Tree Position	**x**			string (250)	The unique four-character alphanumeric code that is created when the address type is defined.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the address type list item was updated.		

1.1.28.3 JContPhone

JContPhone contains information about the phone numbers in a contact record. This information includes the number itself, phone number type (such as Home or Business), and contact record to

which each number belongs. This table represents the **Phone Numbers** block on the **General** tab of a contact record.

Object: JContPhone (J_CONT_PHONE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
number **Before TeamConnect 3.3 SP2: phoneString**	**PHONE_STRING** **(VARCHAR2) (50)**	**Number**	**x**			string (50)	The actual phone number.		**x**
owner-->	**CONTACT_ID** **(NUMBER)**		**x**	**x**	TContact	object	8 The contact to which the phone number belongs. --> Links to specific information for the contact record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the phone number record.		
type-->	**TYPE_ID** **(NUMBER)**	**Type**	**x**	**x**	LContPhoneType	object	8 The type of this phone number (such as Home or Business). --> Links to the table that contains all phone number types.	**x**	**x**
version	**VERSION**		**x**			int	Indicates how many times		

	(NUMBER)						the phone number was updated.		

1.1.28.3.1 LContPhoneType

LContPhoneType contains information about the phone number types defined for contact phone numbers. This information includes the name of the phone type, tree position, and display order. This table represents the **Phone Type** system lookup table.

Object: LContPhoneType (L_CONT_PHONE_TYPE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	**Temp/ Wiz**
childList-->	**n/a**						Do not apply. These lookup items cannot have a hierarchical structure.		
parent-->	**PARENT_ID (NUMBER)**								
displayOrder	**DISPLAY_ORDER (NUMBER)**	**Order**	**x**			int	The order in which the table items will be listed in the GUI, lowest number first.		
name	**NAME (VARCHAR2) (50)**	**Item Name**	**x**			string (50)	The name of the phone type.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the phone type list item.		
treePosition	**TREE_POSITION (VARCHAR20) (250)**	**Tree Position**	**x**			string (250)	The unique four-character alphanumeric code that is created when the phone type is defined.		

version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the phone type list item was updated.		

1.1.28.4 JContFax

JContFax contains information about the fax numbers in a contact record. This information includes the number itself, fax type (such as Home or Business), and contact record to which each number belongs. This table represents the Fax Numbers block on the General tab of a contact record.

Object: JContFax (J_CONT_FAX)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
owner-->	**CONTACT_ID (NUMBER)**		**x**	**x**	TContact	object	8 The contact to which the fax number belongs. --> Links to specific information about the contact record.		
number **Before TeamConnect 3.3 SP2: faxString**	**FAX_STRING (VARCHAR2) (50)**	**Number**	**x**			string (50)	The actual fax number.		**x**
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the fax number record.		
type-->	**TYPE (NUMBER)**	**Type**	**x**	**x**	LContFaxType	object	8 The type of the fax number. --> Links to definition information for the fax type.	**x**	**x**

version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the fax number was updated.		

1.1.28.4.1 LContFaxType

LContFaxType contains information about the fax types defined for contact fax numbers including the name of the fax type, tree position, and display order. This table represents the **Fax Type** system lookup table.

Object: LContFaxType (L_CONT_FAX_TYPE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
childList-->	**n/a**						Do not apply. These lookup items cannot have a hierarchical structure.		
parent-->	**PARENT_ID (NUMBER)**								
displayOrder	**DISPLAY_ORDER (NUMBER)**	**Order**	**x**			int	The order in which the table items will be listed in the GUI, lowest number first.		
name	**NAME (VARCHAR2) (50)**	**Item Name**	**x**			string (50)	The name of the fax type.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the fax type list item.		

treePosition	**TREE_POSITION** **(VARCHAR2) (250)**	**Tree Position**	**x**			string (250)	The unique four-character alphanumeric code that is created when the fax type is defined.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the fax type list item was updated.		

1.1.28.5 JContEmail

JContEmail contains information about the email addresses in a contact record. This information includes the address itself, email address type (such as Personal or Business), and contact record to which each email address belongs. This table represents the **Email Addresses** block on the **General** tab of a contact record.

Object: JContEmail (J_CONT_EMAIL)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
address **Before TeamConnect 3.3 SP2: emailString**	**EMAIL_STRING** **(VARCHAR20) (250)**	**Email**	**x**			string (250)	The actual email address.		**x**
owner-->	**CONTACT_ID** **(NUMBER)**		**x**	**x**	TContact	object	8 The contact to which the email address belongs. --> Links to specific information for the contact record.		

primary Key	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the email address record.		
type-->	**TYPE_ID (NUMBER)**	**Type**	**x**	**x**	LContEmailType	object	8 The type of the email address. --> Links to definition information for the email address type.	**x**	**x**
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the email address was updated.		

1.1.28.5.1 LContEmailType

LContEmailType contains information about the email types defined for contact email addresses. This information includes the name of the email type, tree position, and display order. This table represents the **Email Type** system lookup table.

Object: LContEmailType (L_CONT_EMAIL_TYPE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
childList-->	**n/a**						Do not apply. These lookup items cannot have a hierarchical structure.		
parent-->	**PARENT_ID (NUMBER)**								
displayOrder	**DISPLAY_ORDER**	**Order**	**x**			int	The order in which the table items will be listed in the GUI, lowest number first.		

	(NUMBER)								
name	**NAME** **(VARCHAR2) (50)**	**Item Name**	**x**			string (50)	The name of the email type.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the email type list item.		
treePosition	**TREE_POSITION** **(VARCHAR20) (250)**	**Tree Position**	**x**			string (250)	The unique four-character alphanumeric code that is created when the email type is defined.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the email type list item was updated.		

1.1.28.6 JContInetAddress

JContInetAddress contains information about the internet addresses in a contact record. This information includes the address itself, internet address type (such as Personal or Business), and contact record to which each address belongs. This table represents the **Internet Addresses** block on the **General** tab of a contact record.

Object: JContInetAddress (J_CONT_INET_ADDRESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
url **Before TeamConnect 3.3 SP2:**	**INET_ADDRESS_STRING** **(VARCHAR2) (250)**	**Address**	**x**			string (250)	The actual internet address.		**x**

inetAddressString									
owner-->	**CONTACT_ID (NUMBER)**		**x**	**x**	TContact	object	8 The contact to which the internet address belongs. --> Links to specific information for the contact record.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the internet address record.		
type-->	**TYPE_ID (NUMBER)**	**Type**	**x**	**x**	LContInetAddressType	object	8 The type of the internet address. --> Links to definition information for the internet address type.	**x**	**x**
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the internet address was updated.		

1.1.28.6.1 LContInetAddressType

LContInetAddressType contains information about the internet address types defined for contact internet addresses. This information includes the name of the internet address type, tree position, and display order. This table represents the **Internet Address Type** system lookup table.

Object: LContInetAddressType (L_CONT_INET_ADDRESS_TYPE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/

									Wiz
childList-->	**n/a**						Do not apply. These lookup items cannot have a hierarchical structure.		
parent-->	**PARENT_ID (NUMBER)**								
displayOrder	**DISPLAY_ORDER (NUMBER)**	**Order**	**x**			int	The order in which the table items will be listed in the GUI, lowest number first.		
name	**NAME (VARCHAR2) (50)**	**Item Name**	**x**			string (50)	The name of the internet address type.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the internet address type list item.		
treePosition	**TREE_POSITION (VARCHAR2) (250)**	**Tree Position**	**x**			string (250)	The unique four-character alphanumeric code that is created when the internet address type is defined.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the internet address type list item was updated.		

1.1.28.7 JContSkill

JContSkill contains information about the skills in a contact record including the skill level, skill type (such as underwriting or litigation), and contact record to which each skill belongs. This table represents the Skills block on the Skills tab of a contact record.

Object: JContSkill (J_CONT_SKILL)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
owner-->	**CONTACT_ID (NUMBER)**		**x**	**x**	TContact	object	8 The contact to which the skill belongs. --> Links to specific information for the contact record.		
skillLevel	**SKILL_LEVEL (NUMBER)**	**Level of Expertise**	**x**			int	The skill level assigned to the contact for the skill type.		**x**
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the skill.		
skillType--> Before TeamConnect 3.3 SP2: type-->	**TYPE_ID (NUMBER)**	**Type**	**x**	**x**	LContSkillType	object	8 The type of the skill. --> Links to definition information for the skill type.	**x**	**x**
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the skill record was updated.		

1.1.28.7.1 LContSkillType

LContSkillType contains information about the skill types defined for contact skills. This information includes the name of the skill, tree position, and display order. This table represents the Skill Type system lookup table.

Object: LContSkillType (L_CONT_SKILL_TYPE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
childList-->	**n/a**		**x**	**x**	LContSkillType		8 The child skill types for the current skill type. --> Links to definition information for a child skill type.		
displayOrder	**DISPLAY_ORDER** **(NUMBER)**	**Order**	**x**			int	The order in which the table items will be listed in the GUI, lowest number first.		
name	**NAME** **(VARCHAR2) (50)**	**Item Name**	**x**			string (50)	The name of the skill type.		
parent-->	**PARENT_ID** **(NUMBER)**	**Show items in node**	**x**	**x**	LContSkillType	object	8 The parent item of the current item. --> Links to definition information for the parent skill type.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the skill type list item.		
treePosition	**TREE_POSITION** **(VARCHAR2) (250)**	**Tree Position**	**x**			string (250)	The unique four-character alphanumeric code that is created when the skill type is defined.		

version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the skill type list item was updated.		

1.1.28.8 JContDefaultRate

JContDefaultRate contains information about the default rate in a contact record. (The default rate is used when there is no specific rate for a task's category.) This information includes the rate amount, effective from and to date, and contact record to which each rate belongs.

Rates that are specific to categories are described in JContRate.

Object: JContDefaultRate (J_CONT_DEFAULT_RATE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
firstEffectiveDate **Before TeamConnect 3.3 SP2: effectiveFrom**	**EFFECTIVE_FROM** **(DATE)**	**From**	**x**			date	The date and time when the rate will begin to apply to the associated task. Time-zone-independent.	**x**	**x**
lastEffectiveDate **Before TeamConnect 3.3 SP2: effectiveTo**	**EFFECTIVE_TO** **(DATE)**	**To**	**x**			date	The date and time when the rate will cease applying to the associated task. Time-zone-independent.	**x**	**x**
owner-->	**CONTACT_ID** **(NUMBER)**		**x**	**x**	TContact	object	8 The contact to which the rate belongs. --> Links to specific information for the contact record.		

primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the rate record.		
rate **Before TeamConnect 3.3 SP2: rateAmount**	**RATE_AMOUNT (NUMBER)**	**Rate**	**x**			decimal	The actual rate for the associated task.	**x**	**x**
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the rate record was updated.		

1.1.28.9 JContRate

JContRate contains information about the rates in a contact record. This information includes the rate amount, effective from and to date, task category, and contact record to which each rate belongs.

This table represents the **Rates** block on the **Rates** page of a contact record. It also represents the **Invoice Task Rates** block on the **Rates** page of a contact record. What determines which block will display a given rate record is the value in the TASK_CATEGORY attribute. If this value links to a category assigned to the **Task** object type, then the rate appeasr in the Rates block. If instead TASK_CATEGORY links to a category from the **Task Categories** lookup table in the **Invoice Line Item** object type, then the rate appears in the Invoice Task Rates block.

Object: JContRate (J_CONT_RATE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
firstEffectiveDate **Before TeamConnect 3.3 SP2: effectiveFrom**	**EFFECTIVE_FROM (DATE)**	**From**	**x**			date	The date and time when the rate will begin to apply to the associated task. Time-zone-independent.	**x**	**x**

lastEffectiveDate **Before TeamConnect 3.3 SP2: effectiveTo**	**EFFECTIVE_TO** **(DATE)**	**To**	**x**			date	The date and time when the rate will cease applying to the associated task. Time-zone-independent.	**x**	**x**
owner-->	**CONTACT_ID** **(NUMBER)**		**x**	**x**	TContact	object	8 The contact to which the rate belongs. --> Links to specific information for the contact record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the rate record.		
rate **Before TeamConnect 3.3 SP2: rateAmount**	**RATE_AMOUNT** **(NUMBER)**	**Rate**	**x**			decimal	The actual rate for the associated task.	**x**	**x**
taskCategory-->	**TASK_CATEGORY** **(NUMBER)**	**Task**	**x**	**x**	YRecentlyViewedRecord	object	8 The task category that belongs to the rate. --> Links to definition information for the task category.	**x**	**x**
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the rate record was updated.		

1.1.28.10 JContRelation

JContRelation contains information about a contact record's relations with other contacts. This information includes the contact on the left and right sides of the relationship and the type of relationship between the two contacts (such as "Attorney for" or "Employee of"). This table represents the **Relations** block on the **Relations** tab of a contact record.

Object: JContRelation (J_CONT_RELATION)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
contact--> Before TeamConnect 3.3 SP2: leftContact-->	**LEFT_CONTACT_ID (NUMBER)**	**Contact** (left-side)	**x**	**x**	TContact	object	8 The contact that is on the left side of the relationship. --> Links to specific information for the contact record. ***Note:*** *The distinction between right and left contact in TeamConnect 3.4 is determined by a 3.4-specific attribute*		**x**

							named ***relationDirection****.*		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the relation record.		
contact--> **Before TeamConnect 3.3 SP2: rightContact -->**	**RIGHT_CONTACT_ID** **(NUMBER)**	**Contact** (right-side)	**x**	**x**	TContact	object	8 The contact that is on the right side of the relationship. --> Links to specific information for the contact record. ***Note:*** *The distinction between right and left contact in TeamConnect 3.4 is determined by a 3.4-specific attribute named* ***relationDirection****.*		**x**

relationType **Before TeamConnect 3.3 SP2: typeID**	**TYPE_ID** **(NUMBER)**	**Relation**	**x**	**x**	LContRelationType	object	8 The type of the relationship. --> Links to definition information for the relation type.	**x**	**x**
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the relation record was updated.		

1.1.28.10.1 LContRelationType

LContRelationType contains information about the relationship types defined for contact records. This information includes the name of the relation type, tree position, and display order. This table represents the **Contact Relation Type** system lookup table.

Object: LContRelationType (L_CONT_RELATION_TYPE)

Attribute	Database column name	Field in UI	End of path	Bridge	Data type	Comments	Commonly used in:	
							Rules	Temp/Wiz
childList-->	**n/a**					Do not apply. These lookup items cannot have a hierarchical structure.		
parent-->	**PARENT_ID** **(NUMBER)**							
displayOrder	**DISPLAY_ORDER** **(NUMBER)**	**Order**	**x**		int	The order in which the table items will be listed in the GUI, lowest number first.		
name	**NAME** **(VARCHAR2) (50)**	**ItemName**	**x**		string (50)	The name of the relation type.		

primaryKey	**PRIMARY_KEY (NUMBER)**		**x**		int	The unique ID of the relation type list item.		
treePosition	**TREE_POSITION (VARCHAR2) (250)**	**Tree Position**	**x**		string (250)	The unique four-character alphanumeric code that is created when the relation type is defined.		
version	**VERSION (NUMBER)**		**x**		int	Indicates how many times the relation type list item was updated.		

1.1.28.11 JContTerritory

JContTerritory contains information about the territories in a contact record. This information includes the territory type (such as Northwest or Southeast), and contact record to which each territory belongs. This table represents the **Territories** block on the **Territories** tab of a contact record.

Object: JContTerritory (J_CONT_TERRITORY)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
owner-->	**CONTACT_ID (NUMBER)**		**x**	**x**	TContact	object	8 The contact to which the territory belongs. --> Links to specific information for the contact record.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the territory record.		
territoryItem-->	**TYPE_ID**	**Territory**	**x**	**x**	LContTerritoryType	object	8 The type of the territory record.	**x**	**x**

Before TeamConnect 3.3 SP2: type-->	**(NUMBER)**						--> Links to definition information for the territory type.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the territory record was updated.		

1.1.28.11.1 LContTerritoryType

LContTerritoryType contains information about the territory types defined for contact records. This information includes the name of the territory type, tree position, and display order. This table represents the **Territory Type** system lookup table.

Object: LContTerritoryType (L_CONT_TERRITORY_TYPE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
childList-->	**n/a**		**x**	**x**	LContTerritoryType		8 The child territories for this territory. --> Links to definition information for all child territories of the current territory.		
displayOrder	**DISPLAY_ORDER** **(NUMBER)**	**Order**	**x**			int	The order in which the table items will be listed in the GUI, lowest number first.		
name	**NAME** **(VARCHAR2) (50)**	**Item Name**	**x**			string (50)	The name of the territory type.		

parent -->	**PARENT_ID (NUMBER)**	**Show items in node**	**x**	**x**	LContTerritoryType	object	8 The parent item of the current item. --> Links to definition information for the parent item.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the territory type list item.		
treePosition	**TREE_POSITION (VARCHAR2) (250)**	**Tree Position**	**x**			string (250)	The unique four-character alphanumeric code that is created when the territory type is defined.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the territory type list item was updated.		

1.1.28.12 Y_Contact_Group

Y_Contact_Group represents contact groups (Address Books) and contains information such as the user who owns the contact group, whether the contact group is public, and name of the contact group. This table represents the **Contact Group Information** block on the **General** tab of a contact group record. The list of members is available through the **memberList** attribute.

Object: Y_Contact_Group

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
createdBy -->	**CREATED_BY_ID**		**x**	**x**	YUser	object	8 User who created the contact group (Address Book).		

	(NUMBER)						--> Links to specific information for the user account.		
createdOn	**CREATED_ON** **(DATE)**		**x**			date	Date when the contact group (Address Book) was created.		
members--> **Before TeamConnect 3.3 SP2: memberList-->**	**n/a**			**x**	JDistMember		Links to the list of members in the contact group (Address Book).		
modifiedBy-->	**MODIFIED_BY_ID** **(NUMBER)**		**x**	**x**	YUser	object	8 User who last modified the contact group (Address Book). --> Links to specific information for the user account.		
modifiedOn	**MODIFIED_ON** **(NUMBER)**		**x**			date	Date when the contact group (Address Book) was last modified.		
name	**NAME** **(VARCHAR2) (250)**	Group name (**General** block of Contact Group)	**x**			string (250)	The name of the contact group (Address Book).		
owner-->	**USER_ID**		**x**	**x**	YUser	object	8 The user to whom the		

	(NUMBER)						contact group (Address Book) belongs. --> Links to specific information about the user account.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the contact group (Address Book).		
shared **Before TeamConnect 3.3 SP2: isPublic**	**IS_PUBLIC** **(NUMBER)**	Is contact group (Address Book) public (**General** block of Contact Group)	**x**			boolean	The setting that indicates whether the contact group is public.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the item was updated.		

1.1.28.12.1 JDistMember

JDistMember contains information about the members of a contact group (Address Book). This information includes the contact who is a member of a contact group and the contact group to which the member belongs. This table represents the **Contact Group Information** block on the **Members** tab of a contact group record.

Object: JDistMember (J_DIST_MEMBER)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz

contact-->	**CONTACT_ID** **(NUMBER)**	**Member** (**Members** block of Contact Group)	**x**	**x**	TContact	object	8 The contact who is a member of a contact group (Address Book). --> Links to specific information for the contact record.		
createdBy-->	**CREATED_BY_ID** **(NUMBER)**		**x**	**x**	YUser	object	8 User who created the contact group (Address Book) member. --> Links to specific information for the user account.		
createdOn	**CREATED_ON** **(DATE)**		**x**			date	Date when the contact group (Address Book) member was created.		
distribution-->	**CONTACT_GROUP_ID** **(NUMBER)**		**x**	**x**	RDistribution	object	8 The contact group (Address Book) to which this contact group member belongs. --> Links to definition information for the contact group.		
modifiedBy-->	**MODIFIED_BY_ID** **(NUMBER)**		**x**	**x**	YUser	object	8 User who last modified the contact group (Address Book) member. --> Links to specific information for the user account.		
modifiedOn	**MODIFIED_ON**		**x**			date	Date when the contact group		

	(DATE)						(Address Book) member was last modified.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the contact group (Address Book) member.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the item was updated.		

1.1.28.13 EContDetail

EContDetail contains all categories added to contact records. Each time a category is added to a contact record, an entry is made in this table in the database.

In Object Navigator, **EContDetail** is not displayed when you traverse using the **detailList-->** bridge. Instead, a list of all categories that have been defined for the object definition appears. This list is labeled **Category**. From each category, you can traverse to a table called **Detail Fields**, which displays all custom fields belonging to the category you selected.

For more information about how custom field values are stored in the database, see Object Model: Custom Fields.

Object: EContDetail (E_CONT_DETAIL)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
category-->	**CATEGORY_ID** **(NUMBER)**	**Category**	**x**	**x**	YRecentlyViewedRecord	object	8 The category definition to which the added category belongs. --> Links to definition information for the category.		

isManual	**IS_MANUAL (NUMBER)**		**x**			boolean	Specifies whether the category is added manually or automatically to the contact. Categories are added automatically when they are the parent of the category that is added manually. 0 - Added Automatically 1 - Added Manually		
owner-->	**ENTERPRISE_OBJECT_ID (NUMBER)**			**x**	TContact	object	The contact with which the custom fields are associated.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the contact category record.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the added category was updated.		

1.1.28.14 EContUserAccess

EContUserAccess contains the user access information that is set in the **Security** block of each contact record. For example, it includes whether each user that is listed in the Security block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EContUserAccess (E_CONT_USER_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz

isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM (NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1		

							(selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(NUMBER)**	**Option**	**x**			Enum	Indicates whether the user is allowed or denied access to the record. a - Allow (ALLOW) d - Deny (DENY)		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the user rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	TContact	object	8 The contact record to which this user access setting belongs. --> Links to specific information for the contact record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the user access rights.		
user-->	**USER_ID** **(NUMBER)**	**List**	**x**	**x**	YUser	object	8 The user the access rights pertain to.		

							--> Links to specific information about the user account.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the user access rights have been updated.		

1.1.28.15 EContGroupAccess

EContGroupAccess contains the user access information that is set in the Security block of each contact record. For example, it includes whether each user that is listed in the Security block is allowed or denied the Read, Update, Delete, and Perm(ission) rights.

Object: EContGroupAccess (E_CONT_GROUP_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the		

							operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM (NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID (NUMBER)**	**Option**	**x**			Enum	Indicates whether the group is allowed or denied access to the record. a - Allow (ALLOW) d - Deny (DENY)		
group-->	**GROUP_ID (NUMBER)**	**List**	**x**	**x**	YGroup	object	8 The group that the access rights pertain to. --> Links to specific information for the group account.		

isManual	**IS_MANUAL (NUMBER)**		**x**			boolean	Indicates whether the group access rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID (NUMBER)**		**x**	**x**	TContact	object	8 The contact record to which this group access setting belongs. --> Links to specific information for the contact record.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	Unique ID for the group access rights.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the group access rights have been updated.		

1.1.29 Object Model: Appointments

The tables in this appendix provide information about the object model as it relates to appointments.

1.1.29.1 TAppointment

TAppointment contains general information for an appointment record. This information includes the appointment's subject, start date and time, end date and time, default category, geographical area, location, and so on. It also links to the lists of attendees, notes, used resources, and so on.

Object: TAppointment (T_APPOINTMENT)

Attribute	Database column name	Field in UI	End of	Bridge	Links to object table:	Data	Comments	Commonly used in:

			path			type		Rules	
arealtem-->	**AREA_ITEM_ID (NUMBER)**	**Area (General** block)	**x**	**x**	LApptArealtem	object	8 Geographical area where the appointment will take place (for example, Orange County, and Chicago). --> Links to the definition information for the appointment area item.		**x**
attendees--> **Before TeamConnect 3.3 SP2: attendeeList-->**	**n/a**	**Attendees** tab	**x**	**x**	JApptAttendee		8 The list of all attendees that are added to the appointment record. --> Links to the join table that contains information for all attendees who have been added to the appointment.		
categories--> **Before TeamConnect 3.3 SP2: detailList-->**	**n/a**		**x**	**x**	EApptDetal In Object Navigator, links to Category list which appears in the UI only.		8 The list of categories added to the current object. --> In the object model, the added categories		

							link to the values that have been added for custom fields, according to field type. --> In Object Navigator, this attribute is enhanced. It links to a list of all categories defined for the object definition, rather than those added to the current object. From there, you can select a category and traverse to the list of custom fields that belong to it.		
createdBy-->	**CREATED_BY_ID** **(NUMBER)**	**Created By** (**Security** block)	**x**	**x**	YUser	object	8 User who created the appointment. --> Links to specific information about the user account.	**x**	**x**
createdOn	**CREATED_ON** **(DATE)**	**Created On** (**Security** block)	**x**			date	Date and time the appointment was created.	**x**	**x**

createdOnBehalfOf	**CREATED_ON_BEHALF_OF_ID**			**x**	TContact	object	When not null, it indicates that the record was created through an e-billing system or other external application. The contact referenced will often be a vendor in TeamConnect.	**X**	
defaultCategory-->	**DEFAULT_CATEGORY_ID (NUMBER)**	**Default Category (General block)**	**x**	**x**	YRecentlyViewedRecord	object	8 Default category for the appointment. --> Links to the definition information of the category.		**x**
documentFolder-->	**DOCUMENT_FOLDER_ID (NUMBER)**	**Documents block**		**x**	TDocument	object	Links to specific information about the document folder.		
endOn	**END_ON (DATE)**	**Ends On**	**x**			date time	Date and time when the appointment ends.	**x**	**x**
groupAccessList-->	**n/a**			**x**	EApptGroupAccess		Links to the group security information entered through the Security block of the record.		

historyList-->	**n/a**	**History** tab	**x**	**x**	THistory		8 The list of all histories that are added to the appointment record. --> Links to history information for all of the appointment's history records.		
iallDay **Before TeamConnect 3.3 SP2: isAllDay**	**IS_ALL_DAY** **(NUMBER)**		**x**			boolean	Indicates whether the appointment is scheduled for the entire day. 0 - not all day 1 - all day	**x**	**x**
location	**LOCATION** **(VARCHAR2) (250)**	**Location**	**x**			string (250)	Location where the appointment takes place (such as Corporate Headquarters or Conference Room 3rd Floor).		**x**
modifiedBy-->	**MODIFIED_BY_ID** **(NUMBER)**	**Modified By** (Security block)	**x**	**x**	YUser	object	8 User who most recently modified the account. --> Links to specific information about the user account.		

modifiedOn	**MODIFIED_ON** **(DATE)**	**Modified On** (Security block)	**x**			date	Date and time the appointment was most recently modified.		
note-->	**NOTE_ID** **(NUMBER)**	**Notes** (**General** block)		**x**	JNote	object	Links to the table that contains the text entered into the **Notes** field of the record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the appointment record.		
project-->	**PROJECT_ID** **(NUMBER)**	**Project** (**General** block)	**x**	**x**	TProject	object	8 Project for which the appointment is created. --> Links to specific information about the project record.	**x**	**x**
resources --> **Before TeamConnect 3.3 SP2: resourceList-->**	**n/a**	**Resources** tab	**x**	**x**	JApptResource		8 The list of all resources that are added to the appointment record. --> Links to the join table that contains information for all resources added to the appointment.		**x**

							In Object Navigator, this attribute links to a table named **Resource Type** that allows you to filter the resources based on a type, if desired.		
securityTypeIID	**SECURITY_TYPE_IID** **(NUMBER)**	**Security** (**Security** block)	**x**			Enum	Specifies whether the appointment is public or private. 0 - Public (PUBLIC) 2 - Private (PRIVATE)		
startOn	**START_ON** **(DATE)**	**Begins On** (**General** block)	**x**			date time	Date and time when the appointment begins.	**x**	**x**
subject	**SUBJECT**	**Subject** (**General** block)	**x**			string (250)	Description of the appointment (e.g. Meeting with client).		**x**
userAccessList-->	**n/a**			**x**	EApptUserAccess		Links to the user security information entered through the Security block of the record.		

version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the appointment record was updated.		

1.1.29.2 LApptArealtem

LApptArealtem contains information about the area items defined for appointment records. This information includes the name of the area item, tree position, and display order. This table represents the **Area Item** system lookup table.

Object: LApptArealtem (L_APPT__AREA_ITEM)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
childList-->	**n/a**			**x**	LApptArealtem		Links to the child items in the Area Item system lookup table that belong to this area list item.		
displayOrder	**DISPLAY_ORDER (NUMBER)**	**Order** (**Lookup Tables**, **System** tab)	**x**			int	The order in which the table items will be listed in the GUI, lowest number first.		
name	**NAME (VARCHAR2) (250)**	**Item Name** (**Lookup Tables**, **System** tab)	**x**			string (50)	The name of the area item (for example, Orange County, Los Angeles).		

parent->	**PARENT_ID** (**NUMBER**)	**Show items in node** (**Lookup Tables**, **System** tab)	**x**	**x**	LApptArealtem	object	Links to the parent item of this area item in the Area Item system lookup table.		
primaryKey	**PRIMARY_KEY(NUMBER)**		**x**			int	The unique ID of the area list item.		
treePosition	**TREE_POSITION** (**VARCHAR2) (250)**	**Tree Position** (**Lookup Tables**, **System** tab)	**x**			string (250)	The unique four-character alphanumeric code that is created when the area list item is defined.		
version	**(VERSION)** (**NUMBER**)		**x**			int	Indicates how many times the area list item was updated.		

1.1.29.3 JApptAttendee

JApptAttendee contains information about the attendees in an appointment record. This information includes the user who has been added as an attendee, attendance status, and starting and ending date and time. This table represents the **Attendees** block on the **Attendees** tab of an appointment.

Object: JApptAttendee (J_APPT_ATTENDEE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz

attendanceType **Before TeamConnect 3.3 SP2: attendanceTypeID**	**ATTENDANCE_TYPE_ID** **(NUMBER)**	**Attendance** (**Attendees** block)	**x**			int	Indicates the attendance status of the attendee: 0 - Will Attend (WILL_ATTEND) 1 - Tentative (TENTATIVE) 2 - Will Not Attend (WILL_NOT_ATTEND) 3 - Unknown		**x**
endOn	**END_ON** **(DATE)**	**End Time** (**Attendees** block)	**x**			date Time	End date and time until which the user is scheduled to attend.		**x**
owner-->	**APPOINTMENT_ID** **(NUMBER)**			**x**	TAppointment	object	The appointment which the user is scheduled to attend.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID given to each attendee added to an appointment record.		
startOn	**START_ON** **(DATE)**	**Start Time** (**Attendees** block)	**x**			date Time	Start date and time when the user is scheduled to attend.		**x**
user-->	**USER_ID** **(NUMBER)**	**Attendee**	**x**	**x**	YUser	object	8 User who is added as an attendee to the appointment record.	**x**	**x**

		(**Attendees** block)					--> Links to specific information about the user account.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the appointment attendee was updated.		

1.1.29.4 JApptResource

JApptResource contains information about the resources used in an appointment record. This information includes resource type (such as Projector or Conference Room) and the starting and ending date and time the resource is needed. This table represents the **Resources** block on the **Resources** tab of an appointment record.

Object: JApptResource (J_APPT_RESOURCE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	**Temp/Wiz**
endOn	**END_ON** **(DATE)**	**To** (**Resources** block)	**x**			date Time	Date and time until which the resource is needed.		**x**
owner -->	**APPOINTMENT_ID** **(NUMBER)**			**x**	TAppointment	object	The appointment for which the resource is needed.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID given to each resource added to an appointment record.		
startOn	**START_ON**	**From**	**x**			date Time	Date and time when the		**x**

	(NUMBER)	(**Resources** block)					resource is needed.		
type-->	**TYPE_ID** **(NUMBER)**	**Resource** (**Resources** block)	**x**	**x**	LApptResourceType	object	8 The type of resource (such as Projector or Conference Room). --> Links to the definition information for the appointment resource type.		**x**
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the appointment resource was updated.		

1.1.29.5 LApptResourceType

LApptResourceType contains information about the resource types defined for appointment records. This information includes the resource name, tree position, and display order. This table represents the **Resource Type** system lookup table.

Object: LApptResourceType (L_APPT_RESOURCE_TYPE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
childList-->	**n/a**			**x**	LApptResourceType		Links to the list of child resource types that belong to this resource type.		
displayOrder	**DISPLAY_ORDER** **(NUMBER)**	**Order**	**x**			int	The order in which the table items will be listed in the GUI, lowest number first.		

name	**NAME** **(VARCHAR2) (50)**	**Item Name**	**x**			string (50)	The name of the resource type (e.g. Projector).		
parent-->	**PARENT_ID** **(NUMBER)**	**Show items in node**	**x**	**x**	LApptResourceType	object	8 The parent of the current resource type. --> Links to the definition information for the appointment resource type.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the resource list item.		
treePosition	**TREE_POSITION** **(VARCHAR2) (250)**	**Tree Position**	**x**			string (250)	The unique four-character alphanumeric code that is created when the resource list item is defined.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the resource type was updated.		

1.1.29.6 EApptDetail

EApptDetail contains all categories added to appointment records. Each time a category is added to an appointment record, an entry is made in this table in the database.

In Object Navigator, **EApptDetail** is not displayed when you traverse using the **detailList-->** bridge. Instead, a list of all categories that have been defined for the object definition appears. This list is labeled **Category**. From each category, you can traverse to a table called **Detail Fields**, which displays all custom fields belonging to the category you selected.

For more information about how custom field values are stored in the database, see Object Model: Custom Fields.

Object: EApptDetail (E_APPT_DETAIL)

Attribute	Database column	Field in UI	End	Bridg	Links to object	Data	Comments	Commonly used

	name		of path	e	table:	type		in:	
								Rules	Temp/Wiz
category-->	CATEGORY_ID (NUMBER)	Category	x	x	YRecentlyViewedRecord	object	The category of the appointment with which the details are associated.		
owner-->	ENTERPRISE_OBJECT_ID (NUMBER)			x	TAppointment	object	The appointment with which the custom fields are associated.		
isManual	IS_MANUAL (NUMBER)		x			boolean	Specifies whether the category is added manually or automatically to the appointment. Categories are added automatically when they are the parent of the category that is added manually. 0 - Added Automatically 1 - Added Manually		
primaryKey	PRIMARY_KEY (NUMBER)		x			int	The unique ID of the appointment category record.		
version	VERSION (NUMBER)		x			int	Indicates how many times the added category was updated.		

1.1.29.7 EApptUserAccess

EApptUserAccess contains the user access information that is set in the **Security** block of each appointment record. For example, this information includes whether each user that is listed in the Security block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EApptUserAccess (E_APPT_USER_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1		

							(selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM** **(NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(NUMBER)**	**Option**	**x**			Enum	Indicates whether the user is allowed or denied access to the record. a - Allow (ALLOW) d - Deny (DENY)		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the user rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		

owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	TAppointment	object	8 The appointment record with which this user access setting is associated. --> Links to the specific information about the appointment record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the user access rights.		
user-->	**USER_ID** **(NUMBER)**	**List**	**x**	**x**	YUser	object	8 Specifies the user the access rights pertain to. --> Links to specific information about the user account.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the user access rights have been updated.		

1.1.29.8 EApptGroupAccess

EApptGroupAccess contains the group access information that is set in the **Security** block of each appointment record. For example, it includes whether each group that is listed in the Security block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EApptGroupAccess (E_APPT_GROUP_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ** **(NUMBER)**	**Read** che	**x**			int	Specifies whether the Read operation has been selected for granting or		

		ck box					denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM (NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine		

							whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(CHAR) (1)**	**Option**	**x**			Enum	Indicates whether the group is allowed or denied access to the record. a - Allow (ALLOW) d - Deny (DENY)		
group->	**GROUP_ID** **(NUMBER)**	**List**	**x**	**x**	YGroup	object	8 Specifies the group that the access rights pertain to. --> Links to specific information about the group account.		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the group access rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	TAppointment	object	8 The appointment record with which this group access setting is associated. --> Links to the specific information about the appointment record.		
primaryKey	**PRIMARY_KEY**		**x**			int	Unique ID for the group access rights.		

	(NUMBER)								
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the group access rights have been updated.		

1.1.30 Object Model: Accounts

The tables in this appendix provide information about the object model as it relates to accounts.

1.1.30.1 TAccount

TAccount contains the general information for an account record including the account's allocation amount and limit, used amount, and activity status. This information also includes expense information, default category, posting criteria, and so on. This table also links to the list of custom fields created for the account categories and other related information contained in other tables.

Object: TAccount (T_ACCOUNT)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
accountInvolvedType **Before TeamConnect 3.3 SP2: applTypeInvolvedIID**	**INVOLVED_APPL_TYPE_IID** **(NUMBER)**	Post Account to Involved (Posting Criteria block)	**x**			Enum	Specifies whether the involved is associated to a specific project or to any project. 1 - Any (ANY) 2 - One (ONE)	**x**	**x**
accountOverdraftType	**OVERDRAFT_TYP**	Overdraft Type	**x**			Enum	Specifies what	**x**	**x**

Before TeamConnect 3.3 SP2: overdraftTypeIID	**E_IID** **(NUMBER)**	(General block)					happens when there are not enough funds to cover the manual withdrawals, transfers, or automatically posted transactions. 1 - Do not allow negative account (NONNEGATIVE) 2 - Allow negative account (ALLOW_NEGATIVE) 3 - Automatic overdraft protection (OVERDRAFT)		
accountProjectType **Before TeamConnect 3.3 SP2: applTypeProjectIID**	**PROJECT_APPL_TYPE_IID** **(NUMBER)**	Post Project to this Account (Posting Criteria block)	**x**			Enum	Indicates whether to post transactions associated to a specific project, any project, or a project of a specific category.	**x**	**x**

							1 - Any (ANY_PROJECT) 2 - One (ONE_PROJECT)) 3 - By Type (BY_TYPE)		
accountVendorType **Before TeamConnect 3.3 SP2: applTypeVendorIID**	**VENDOR_APPL_TYPE_IID** **(NUMBER)**	Post Account to Vendor (Posting Criteria block)	**x**			Enum	Indicates whether to post transactions associated to a specific vendor or any vendor. 1 - Any (ANY_VENDOR) 2 - One (ONE_VENDOR)	**x**	**x**
allocated	**TOTAL_ALLOCATED** **(NUMBER)**	Total Allocated (General block)	**x**			decimal	Total amount of money allocated to the account through deposits or automatic transfers.	**x**	**x**
allocationLimit	**ALLOC_LIMIT** **(NUMBER)**	Allocation Limit (General block)	**x**			decimal	Total amount of funds allowed to be allocated to the account records as specified	**x**	**x**

							by the user.		
accrual	**TOTAL_ACCRUAL** **(NUMBER)**	Total Accruals (General block)	**x**			decimal	Total amount of accrual invoices that have been posted to this account, to date. Takes into account voids of accrual invoices.	**x**	**x**
applProjDetailField-->	**APPL_PROJECT_DETAIL_FIELD_ID** **(NUMBER)**						Currently not used.		
autoPost **Before TeamConnect 3.3 SP2: isAutoPost**	**AUTO_POST** **(NUMBER)**	Allow Posting (General block)	**x**			boolean	Indicates whether expense, task, and invoice line item transactions can be posted to the account. 0 - Do not post 1 - Post	**x**	**x**
available	**TOTAL_BG_AVAILABLE**	Balance (General block)	**x**			decimal	Account balance, which results from the		

	(NUMBER)						following equation: allocated - used = available		
categories--> **Before TeamConnect 3.3 SP2: detailList-->**	**n/a**		**x**	**x**	EAcctDetail In Object Navigator, links to Category list which appears in the UI only.		8 The list of categories added to the current object. --> In the object model, the added categories link to the values that have been added for custom fields, according to field type. --> In Object Navigator, this attribute is enhanced. It links to a list of all categories defined for the object definition, rather than those added to the current object. From there, you can select a category and traverse to		

							the list of custom fields that belong to it.		
createdBy-->	**CREATED_BY_ID** **(NUMBER)**	Created By (Security block)	**x**	**x**	YUser	object	8 User who created the account. --> Links to specific information about the user account.		
createdOn	**CREATED_ON** **(DATE)**	Created On (Security block)	**x**			date	Date and time the account was created.		**x**
createdOnBehalfOf	**CREATED_ON_BEHALF_OF_ID**			**x**	TContact	object	When not null, it indicates that the record was created through an e-billing system or other external application. The contact referenced will often be a vendor in TeamConnect.	**X**	
defaultCategory-->	**DEFAULT_CATEGORY_ID**	Default Category (indicated by a	**x**	**x**	YRecentlyViewedRecord	object	8 Default category for the account.		

	(NUMBER)	blue diamond - Categories block)					--> Links to the definition information of the category.		
documentFolder-->	**DOCUMENT_FOLDER_ID** **(NUMBER)**	Documents block		**x**	TDocument	object	Links to specific information about the document folder.		
EndOn **Before TeamConnect 3.3 SP2: applEndOn**	**APPL_END_ON** **(DATE)**	Period ends on: (General block)	**x**			date	Specifies the date after which transactions cannot be posted to the account.	**x**	**x**
expenseCategory--> Before TeamConnect 3.3 SP2: applExpeCategory-->	**APPL_EXPENSE_CATEGORY_ID** **(NUMBER)**	Post Expenses of Category (Posting Criteria block)	**x**	**x**	YRecentlyViewedRecord	object	8 Expense category specified as the posting criterion for the account. --> Links to the definition information of the category.	**x**	**x**
expensePercent **Before TeamConnect 3.3 SP2: applExpePercent**	**APPL_EXPENSE_PERCENT** **(NUMBER**	Expense Percent (Posting Criteria block)	**x**			decimal	Percentage of the total expense amount allowed to be posted to the account.	**x**	**x**

groupAccessList-->	**n/a**			**x**	EAcctGroupAccess		Links to the group security information entered through the Security block of the record.		
historyList-->	**n/a**	History tab	**x**	**x**	THistory		8 The list of all histories that are added to the account record. --> Links to specific history information for all of the account's history records.	**x**	
active **Before TeamConnect 3.3 SP2: iisActive**	**STATUS_IID** **(NUMBER)**	Status (General block)	**x**			boolean	Indicates whether the account is currently active or inactive. 0 - Inactive 1 - Active	**x**	**x**
allowExpense **Before TeamConnect 3.3 SP2: iisApplExpense**	**APPL_EXPENSE** **(NUMBER)**	Post Expenses to this Account (Posting Criteria block)	**x**			boolean	Indicates whether expense transactions can be posted to the account. 0 - Do not post 1 - Post	**x**	**x**

allowInvoiceExpense **Before TeamConnect 3.3 SP2: iisApplLineExpense**	**APPL_INVD_EXPENSE** **(NUMBER)**	Post Invoice Expenses to this Account (Posting Criteria block)	**x**			boolean	Indicates whether invoice line item transactions of type Expense can be posted to the account. 0 - Do not post 1 - Post	**x**	**x**
iallowInvoiceTask **Before TeamConnect 3.3 SP2: isApplLineTask**	**APPL_INVD_TASK** **(NUMBER)**	Post Invoice Tasks to this Account (Posting Criteria block)	**x**			boolean	Indicates whether invoice line item transactions of type Task can be posted to the account. 0 - Do not post 1 - Post	**x**	**x**
allowTask **Before TeamConnect 3.3 SP2: iisApplTask**	**APPL_TASK** **(NUMBER)**	Post Tasks to this Account (Posting Criteria block)	**x**			boolean	Indicates whether task transactions can be posted to the account. 0 - Do not post 1 - Post	**x**	**x**
invoiceExpenseCategory--> **Before TeamConnect**	**APPL_INVD_EXP_CATEGORY_ID**	Post Invoice Expense of	**x**	**x**	YRecentlyViewedRecord	object	8 Expense category for an invoice line item that is	**x**	**x**

3.3 SP2: applLineExpeCategory-->	**(NUMBER)**	Category (Posting Criteria block)					specified as the posting criterion for the account. --> Links to the definition information of the category.		
invoiceExpensePercent **Before TeamConnect 3.3 SP2: applLineExpePercent**	**APPL_INVD_EXPENSE_PERCENT** **(NUMBER)**	Invoice Expense Percent (Posting Criteria block)	**x**			decimal	Percentage of the total expense line item amount allowed to be posted to the account.	**x**	**x**
invoiceTaskCategory **Before TeamConnect 3.3 SP2: applLineTaskCategory-->**	**APPL_INVD_TASK_CATEGORY_ID** **(NUMBER)**	Post Invoice Tasks of Type (Posting Criteria block)	**x**	**x**	YRecentlyViewedRecord	object	8 Task category for an invoice line item that is specified as the posting criterion for the account. --> Links to the definition information of the category.	**x**	**x**
invoiceTaskPercent **Before TeamConnect 3.3 SP2:**	**APPL_INVD_TASK_PERCENT** **(NUMBER)**	Invoice Task Percent (Posting Criteria block)	**x**			decimal	Percent of the task invoice line item allowed to be posted	**x**	**x**

applLineTask Percent							to the account. Note: This column is populated only when **allowInvoiceTask** is set to 1.		
applInvcNonUSTaxType-->	**APPL_NON_US_TAX_TYPE** **(NUMBER)**	Non-US Tax Type		**x**	LInvcNonUSTaxType	object	8 The tax type category associated with the tax amount.	**x**	**x**
applInvcNonUSTaxPercent	**APPL_INVC_NON_US_TAX_PERCENT** **(NUMBER**	Non-US Tax Percent (Posting Criteria block)	**x**			decimal	Percentage of the total non-US tax amount allowed to be posted to the account.	**x**	**x**
involved--> **Before TeamConnect 3.3 SP2: applInvolved-->**	**APPL_INVOLVED_ID** **(NUMBER)**	This Involved (Posting Criteria block)	**x**	**x**	TContact	object	8 The contact involved in the project who is specified as the posting criterion for the account. --> Links to the specific information about the contact object.	**x**	**x**
modifiedBy-->	**MODIFIED_BY_ID**	Modified By	**x**	**x**	YUser	object	8 User who most recently		

	(NUMBER)	(Security block)					modified the account. --> Links to specific information about the user account.		
modifiedOn	**MODIFIED_ON** **(DATE)**	Modified On (Security block)	**x**			date	Date and time the account was most recently modified.		
name	**NAME** **(VARCHAR2) (250)**	Name (General block)	**x**			string (250)	Name used to describe the account.		**x**
note-->	**NOTE_ID** **(NUMBER)**	Notes (General block)		**x**	JNote	object	Links to the table that contains the text entered into the **Notes** field of the record.		
parentAccount-->	**PARENT_ACCOUNT_ID** **(NUMBER)**	Parent Account (General block)	**x**	**x**	TAccount	object	8 The account that is specified as the parent of this account. --> Links to specific information about the account record.	**x**	**x**

primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the account record.		
project **Before TeamConnect 3.3 SP2: applProject**	**PROJECT_ID** **(NUMBER)**	This Project (Posting Criteria tab)	**x**	**x**	TProject	object	8 Project that is specified as the posting criterion for the account. --> Links to specific information about the project record.	**x**	**x**
projectCategory--> **Before TeamConnect 3.3 SP2: applProjCategory-->**	**APPL_PROJECT_CATEGORY_ID** **(NUMBER)**	Post Projects of Type (Posting Criteria block)	**x**	**x**	YRecentlyViewedRecord	object	8 Project category specified as the posting criterion for the account. --> Links to the definition information of the category.	**x**	**x**
securityTypeID	**SECURITY_TYPEID** **(NUMBER)**	Security Type (Security block)	**x**			int	Specifies whether the account is public or private. 0 - Public (PUBLIC) 2 - Private (PRIVATE)		

startOn **Before TeamConnect 3.3 SP2: applStartOn**	**APP_START_ON** **(DATE)**	Account period begins: (General block)	**x**			date	Specifies the start date from which transactions can be posted to the account.	**x**	**x**
taskCategory--> **Before TeamConnect 3.3 SP2: applTaskCategory-->**	**APPL_TASK_CATEGORY_ID** **(NUMBER)**	Post Tasks of Type (Posting Criteria block)	**x**	**x**	YRecentlyViewedRecord	object	8 Task category specified as the posting criterion for the account. --> Links to the definition information of the category.	**x**	**x**
taskPercent **Before TeamConnect 3.3 SP2: applTaskPercent**	**APPL_TASK_PERCENT** **(NUMBER)**	Task Percent (Posting Criteria block)	**x**			decimal	Percentage of the total task amount allowed to be posted to the account.	**x**	**x**
iisApplInvcStandard	**APPL_INVC_STANDAR D** **(NUMBER)**	Post Invoice Type	**x**			boolean	Indicates whether standard invoices can be posted to the account. 0 - No 1 - Yes	**x**	**x**
iisApplInvcAccrual	**APPL_INVC_ACC**	Post Invoice	**x**			boolean	Indicates whether	**x**	**x**

	RUAL **(NUMBER)**	Type					accrual invoices can be posted to the account. 0 - No 1 - Yes		
iisApplInvcCreditNote	**APPL_INVC_CREDIT_NOTE** **(NUMBER)**	Post Invoice Type	**x**			boolean	Indicates whether credit notes can be posted to the account. 0 - No 1 - Yes	**x**	**x**
iisApplInvcShadow	**APPL_INVC_SHADOW** **(NUMBER)**	Post Invoice Type	**x**			boolean	Indicates whether shadow invoices can be posted to the account. 0 - No 1 - Yes	**x**	**x**
iisApplInvcNonUSTax	**APPL_NON_US_TAX** **(NUMBER)**	Post Non-US Tax	**x**			boolean	Indicates whether non-US tax amounts can be posted to the account. 0 - No 1 - Yes	**x**	**x**
treeKey	**TREE_KEY**		**x**			string	The chain of primary keys that		

	(VARCHAR2) (700)					(700)	identifies the parent tree of the account. Used for searching and reporting purposes only.		
type **Before TeamConnect 3.3 SP2: typeIID**	**TYPE_IID** **(NUMBER)**	Account Type (General block)	**x**			Enum	Indicates the account type. 1 - Budget (BUDGET) 2 - Reserve (RESERVE)	**x**	**x**
used	**TOTAL_USED** **(NUMBER)**	Total Used (General block)	**x**			decimal	Total amount of money used through transfers, withdrawals, and posted transactions.	**x**	
userAccessList-->	**n/a**			**x**	EAcctUserAccess		Links to the user security information entered through the Security block of the record.		
vendor **Before TeamConnect**	**APPL_VENDOR_ID**	This Vendor	**x**	**x**	TContact	object	8 Contact specified as the posting criterion for	**x**	**x**

3.3 SP2: applVendor	**(NUMBER)**	(Posting Criteria block)					the account. --> Links to the specific information about the contact object.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the account record was updated.		
postedType	**POSTED_TYPE_ID** **(NUMBER)**		**x**			int	Indicates which types of transactions have previously posted to this account. 0 - None (NONE_POSTED) 1 - Non-shadow (NON_SHADOW_POSTED) 2 - Shadow invoices (SHADOW_POSTED)	**x**	**x**

1.1.30.2 JTranDetail

JTranDetail contains information about the transactions that have been posted against an account record. This information includes the account to which the transaction belongs, a short description, total amount, and whether the amount was deposited or withdrawn. It also links to RTransaction, which contains the transaction to which the transaction detail belongs. This table represents the Transactions block on the Transactions tab of an account record.

Object: JTranDetail (J_TRAN_DETAIL)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
account-->	**ACCOUNT_ID (NUMBER)**	Account (**Transaction** block of Expense, Invoice, or Task)	**x**	**x**	TAccount	object	8 The account to which the transaction detail belongs. --> Links to specific information about the account record.		
objectID	**OBJECT_ID (NUMBER)**	Used to display the link to the associated record in the **Transactions** block.	**x**			int	The primary key of the record (such as an account, expense, or task record) with which the transaction detail is associated.		
owner-->	**ACCOUNT_TRANS_ID (NUMBER)**			**x**	RTransaction	object	The transaction to which this transaction detail belongs.		
primary Key	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the transaction detail.		
shortDescription	**SHORT_DESCRIPTION (VARCHAR2) (250)**	**Name** (**Transactions** block, Account)	**x**			string (250)	Specifies the name of the financial transaction. Depending on the object, the		

							short description is equal to one of the following: • Description of the transfer, deposit, withdrawal, or expense • Task subject • Invoice number and line item type (task or expense) if it is posted		
statusTypeIID	**STATUS_TYPE_IID** **(NUMBER)**		**x**			int	Indicates whether the amount was deposited to the account or withdrawn from the account. 1 - Deposit 2 - Withdraw		
totalAmount	**TOTAL_AMOUNT** **(NUMBER)**	One of the following: • **Amount** (**Transactions** block of Expense, Invoice, or Task)	**x**			decimal	The total amount that was deposited, withdrawn, transferred, posted or voided from/to the account.		

		• **Debit / Credit** (**Account Transactions** block)							
uniqueCode	**TRANS_UNIQUE_CODE** **(VARCHAR2) (4)**	Used in conjunction with objectID to display the link to the associated record in the **Transactions** block.	**x**			string (4)	The 4-character unique code of the object to which the record associated with the transaction detail belongs.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the transaction detail was updated.		

1.1.30.3 RTransaction

RTransaction contains information about instances of complete financial transactions, usually comprised of smaller transactions across multiple accounts. This information includes the user who created the transaction and the transaction date. It also links to JTranDetail, which contains transaction details for particular accounts. This table represents the **Transaction** tab of an invoice record, which displays the entire transaction.

Object: RTransaction (R_TRANSACTION)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/

									Wiz
createdBy-->	**CREATED_BY_ID** **(NUMBER)**		**x**	**x**	YUser	object	8 The user who created the financial transaction. --> Links to specific information about the user account.		
createdOn	**CREATED_ON** **(DATE)**	**Created On** (**Transactions** block of Expense, Task or Invoice)	**x**			date	The date the transaction was created.		
modifiedBy-->	**MODIFIED_BY** **(NUMBER)**		**x**	**x**	YUser	object	8 The user who voided the expense, task, or invoice. --> Links to specific information about the user account.		
modifiedOn	**MODIFIED_ON** **(DATE)**		**x**			date	The date the expense, task, or invoice was voided.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the financial transaction.		
shortDescription	**SHORT_DESCRIPTION** **(VARCHAR2) (250)**						Currently not used.		

transaction Date	**TRANSACTION_DATE (DATE)**	**Date** (**Transactions** block in Account)	**x**			date	The date the transaction took place.		
transaction DetailList--->	**n/a**		**x**	**x**	JTran Detail		Links to the join table that contains information for all transaction details of this financial transaction.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the transaction has been updated. Note: The version is updated only when financial transactions are voided or re-posted.		

1.1.30.4 EAcctDetail

EAcctDetail contains all categories added to account records. Each time a category is added to an account record, an entry is made in this table in the database.

In Object Navigator, **EAcctDetail** is not displayed when you traverse using the **detailList-->** bridge. Instead, a list of all categories that have been defined for the object definition appears. This list is labeled **Category**. From each category, you can traverse to a table called **Detail Fields**, which displays all custom fields belonging to the category you selected.

For more information about how custom field values are stored in the database, see Object Model: Custom Fields.

Object: EAcctDetail (E_ACCT_DETAIL)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments

category-->	**CATEGORY_ID**	**Category**	**x**	**x**	YRecentlyViewedRecord	object	8 The category of the account with which the details are associated. --> Links to the definition information of the category.
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Specifies whether the category is added manually or automatically to the account. Categories are added automatically when they are the parent of the category that is added manually. 0 - Added Automatically 1 - Added Manually
owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**			**x**	TAccount	object	The account with which the custom fields are associated.
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the account category record.
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the added category was updated.

1.1.30.5 EAcctUserAccess

EAcctUserAccess contains the user access information that is set in the **Security** block of each account record. For example, it includes whether each user that is listed in the Security block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EAcctUserAccess (E_ACCT_USER_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz

isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM (NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected).		

							This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(CHAR) (1)**	**Option**	**x**			char	Indicates whether the user is allowed or denied access to the record for purposes of the operations described in isRead, isUpdate, isDelete, and isPerm. a - Allow (ALLOW) d - Deny (DENY)		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the user rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	TAccount	object	8 The account record with which this user access setting is associated. --> Links to the specific information about the account record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the user access rights.		
user-->	**USER_ID**	**List**	**x**	**x**	YUser	object	8 Specifies the user the access rights		

	(NUMBER)						pertain to. --> Links to the specific information about the user account.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the user access rights have been updated.		

1.1.30.6 EAcctGroupAccess

EAcctGroupAccess contains the group access information that is set in the **Security** block of each account record. For example, it includes whether each group that is listed in the Security block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EAcctGroupAccess (E_ACCT_GROUP_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ** **(NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE** **(NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected).		

							This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM (NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID (CHAR) (1)**	**Option**	**x**			char	Indicates whether the group is allowed or denied access to the record. a - Allow (ALLOW) d - Deny (DENY)		
group-->	**GROUP_ID (NUMBER)**	**List**	**x**	**x**	YGroup	object	8 Specifies the group that the access rights pertain to.		

							--> Links to the specific information about the group account.		
isManual	**IS_MANUAL (NUMBER)**		**x**			boolean	Indicates whether the group access rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID (NUMBER)**		**x**	**x**	TAccount	object	8 The account record with which this group access setting is associated. --> Links to the specific information about the account record.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	Unique ID for the group access rights.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the group access rights have been updated.		

1.1.31 Object Model: Expenses

The tables in this appendix provide information about the object model as it relates to expenses.

1.1.31.1 TExpense

TExpense contains the general information for an expense record. This information includes the expense's date, the user who made the expense, unit price of the item or service, quantity, total amount, posting status, description, and so on. This table links to the list of custom fields created for the expense categories and other related information contained in other tables.

Object: Expense (T_EXPENSE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
categories--> Before TeamConnect 3.3 SP2: detailList -->	**n/a**		**x**	**x**	EExpeDetail In Object Navigator, links to **Category** list which appears in the UI only.		8 The list of categories added to the current object. --> In the object model, the added categories link to the values that have been added for custom fields, according to field type. --> In Object Navigator, this attribute is enhanced. It links to a list of all categories defined for the object definition, rather than those added to the current object. From there, you can select a category and traverse to the list of custom		

							fields that belong to it.		
contact--->	**CONTACT_ID** (NUMBER)	**Contact** (**General** block)	**x**	**x**	TContact	object	8 Vendor who is associated with the expense. --> Links to the specific information about the contact object.	**x**	**x**
createdBy-->	**CREATED_BY_ID** (NUMBER)	**Created By** (**Security** block)	**x**	**x**	YUser	object	8 User who created the expense record. --> Links to specific information about the user account.	**x**	**x**
createdOn	**CREATED_ON** (DATE)	**Created On** (**Security** block)	**x**			date	Date and time the expense record was created.		**x**
createdOnBehalfOf	**CREATED_ON_BEHALF_OF_ID**			**x**	TContact	object	When not null, it indicates that the record was created through an e-billing system or other external application. The contact referenced will often be a vendor in TeamConnect.	**X**	
defaultCategory--->	**DEFAULT_CATEGORY_ID**	**Default Category**	**x**	**x**	YRecentlyViewedRecord	object	8 Default expense category.		

	(NUMBER)	(General block)					--> Links to the definition information of the category.		
documentFolder-->	**DOCUMENT_FOLDER_ID** **(NUMBER)**	**Documents** block		**x**	TDocument	object	Links to specific information about the document folder.		
expenseDate	**EXPENSE_DATE** **(DATE)**	**Expensed On** (**General** block)	**x**			date	Date the expense was incurred. Time-zone-independent.	**x**	**x**
expensedBy-->	**EXPENSED_BY_ID** **(NUMBER)**	**Expensed By** (**General** block)	**x**	**x**	YUser	object	8 User who is responsible for the expenditure. This is not necessarily the user who created the expense record. --> Links to specific information about the user account.	**x**	**x**
groupAccessList-->	**n/a**			**x**	EExpeGroupAccess		Links to the group security information entered through the Security block of the record.		
historyList-->	**n/a**	**History** tab	**x**	**x**	THistory		8 The list of all histories that are related to		

							the expense record. --> Links to specific history information for all of the expense's history records.		
modifiedBy-->	**MODIFIED_BY_ID** **(NUMBER)**	**Modified By** **Security** block)	**x**	**x**	YUser	object	8 User who most recently modified the expense. --> Links to specific information about the user account.		
modifiedOn	**MODIFIED_ON** **(DATE)**	**Modified On** (**Security** block)	**x**			date	Date and time the expense record was most recently modified.		
note-->	**NOTE_ID** **(NUMBER)**	**Notes** (**General** block)		**x**	JNote	object	Links to the table that contains the text entered into the **Notes** field of the record.		
postingStatus **Before TeamConnect 3.3 SP2: postingStatusIID**	**STATUS_IID** **(NUMBER)**	**Posting Status** (**General** block)	**x**			int	Specifies whether the expense was posted or not: 1 - Not Posted 2 - Posted 3 - Failed 4 - Posted For Approval	**x**	
primaryKey	**PRIMARY_KEY**		**x**			int	The unique ID of the expense		

	(NUMBER)						record.		
project-->	**PROJECT_ID** **(NUMBER)**	**Project** (**General** block)	**x**	**x**	TProject	object	8 The project for which the expense was incurred. --> Links to specific information about the project record.	**x**	**x**
quantity	**QUANTITY** **(NUMBER)**	**Quantity** (**General** block)	**x**			decimal	Either the total number of units bought or the total length of time spent to complete the service (Hours.Minutes).	**x**	**x**
securityTypeIID	**SECURITY_TYPE_IID** **(NUMBER)**	**Security Type** (**Security** block)	**x**			int	Specifies whether the expense is public or private. 0 - Public 2 - Private		
shortDescription	**SHORT_DESCRIPTION** **(VARCHAR2) (250)**	**Description** (**General** block)	**x**			string (250)	Description of the expense.		**x**
totalAmount	**TOTAL_AMOUNT** **(NUMBER)**	**Total** (**General** block)	**x**			decimal	Total amount of money spent for the expense that is the product of the price per unit/ hour (**unitPrice**) and the number of	**x**	**x**

							units/hours (**quantity**).		
transaction-->	**TRANSACTION_ID** **(NUMBER)**	**Transactions** block		**x**	RTransaction	object	Links to the transactions associated with the expense record.		
unitPrice	**UNIT_PRICE** **(NUMBER)**	**Unit Price** (**General** block)	**x**			decimal	Either the price per unit or the price per hour of service.	**x**	**x**
userAccessList-->	**n/a**			**x**	EExpeUserAccess		Links to the user security information entered through the Security block of the record.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the expense record was updated.		

1.1.31.2 EExpeDetail

EExpeDetail contains all categories added to expense records. Each time a category is added to an expense record, an entry is made in this table.

In Object Navigator, each table **Category** displays only categories belonging to the selected object. From each category, you can traverse to a table called **Detail Fields**, which displays only custom fields belonging to the category you selected.

For more information about how custom field values are stored in the database, see Object Model: Custom Fields.

Object: EExpeDetail (E_EXPE_DETAIL)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp

									/ Wiz
category-->	**CATEGORY_ID** **(NUMBER)**	**Category**	**x**	**x**	YRecentlyViewedRecord	object	8 The category of the expense with which the details are associated. --> Links to the definition information of the category.		
owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**			**x**	Expense	object	The expense with which the custom fields are associated.		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Specifies whether the category is added manually or automatically to the expense. Categories are added automatically when they are the parent of the category that is added manually. 0 - Added Automatically 1 - Added Manually		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the expense category record.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the added category was updated.		

1.1.31.3 EExpeUserAccess

EExpeUserAccess contains the user access information that is set in the **Security** block of each expense record. For example, it includes whether each user that is listed in the Security block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EExpeUserAccess (E_EXPE_USER_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed		

							or denied. (See allowDenyIID.)		
isPerm	**IS_PERM** **(NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(NUMBER)**	**Option**	**x**			char	Indicates whether the user is allowed or denied access to the record. a - Allow d - Deny		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the user rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	Expense	object	8 The expense record with which this user access setting is associated. --> Links to the specific information about the expense record.		
primaryKey	**PRIMARY_KEY**		**x**			int	Unique ID for the user access rights.		

	(NUMBER)								
user-->	**USER_ID** **(NUMBER**	**List**	**x**	**x**	YUser	object	8 Specifies the user the access rights pertain to. --> Links to the specific information about the user account.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the user access rights have been updated.		

1.1.31.4 EExpeGroupAccess

EExpeGroupAccess contains the group access information that is set in the **Security** block of each expense record. For example, it includes whether each group that is listed in the Security block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EExpeGroupAccess (E_EXPE_GROUP_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ** **(NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE** **(NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial.		

							Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE** **(NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM** **(NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(CHAR) (1)**	**Option**	**x**			char	Indicates whether the group is allowed or denied access to the record. a - Allow d - Deny		
group-->	**GROUP_ID** **(NUMBER)**	**List**	**x**	**x**	YGroup	object	8 Specifies the group that the access rights pertain to.		

							--> Links to the specific information about the group account.		
isManual	**IS_MANUAL (NUMBER)**		**x**			boolean	Indicates whether the group access rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID (NUMBER)**		**x**	**x**	Expense	object	8 The record with which this group access setting is associated. --> Links to the specific information about the expense record.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	Unique ID for the group access rights.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the group access rights have been updated.		

1.1.32 Object Model: Tasks

The tables in this appendix provide information about the object model as it relates to tasks.

1.1.32.1 TTask

TTask contains the general information for a task record including the task's due date, estimated hours, rate amount, start-on date, and completed date. This table links to the list of custom fields created for the task categories and other related information contained in other tables.

Object: TTask (T_TASK)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
acceptanceIID	**ACCEPTANCE_IID** **(NUMBER)**						Currently not used.		
activityItem -->	**ACTIVITY_ITEM_ID** **(NUMBER)**	**Activity** (**General** block)	**x**	**x**	LTaskActivityItem	object	8 Activity item specified for the task. --> Links to the definition information for the task activity item.	**x**	**x**
actualHours	**ACTUAL_HOURS** **(NUMBER)**	**Actual Work** (**General** block)	**x**			decimal	Actual length of time spent to complete the task (Hours.Minutes).	**x**	
assigneeHistory--> Before TeamConnect 3.3 SP2: assigneeList-->	**n/a**			**x**	JTaskAssignee		Links to the join table that contains information for all assignees added to the task.	**x**	**x**
completedDate Before TeamConnect 3.3 SP2:	**COMPLETED_ON** **(DATE)**	**Completed On**	**x**			date	Date and time the task was completed. Time-zone-independent.	**x**	

completedOn		(**General** block)							
completedPercent	**COMPLETED_PERCENT** **(NUMBER)**	**% Complete** (General block)	**x**			decimal	Percentage of the task that is currently complete.	**x**	
contact-->	**CONTACT_ID** **(NUMBER)**	**Contact** (**General** block)	**x**	**x**	TContact	object	8 Contact who is associated to the task, e.g. for whom the task is performed. --> Links to the specific information about the contact object.		**x**
createdBy-->	**CREATED_BY_ID** **(NUMBER)**	**Created By** (**Security** block)	**x**	**x**	YUser	object	8 User who created the task record. --> Links to specific information about the user account.	**x**	**x**
createdOn	**CREATED_ON** **(DATE)**	**Created On** (**Security** block)	**x**			date	Date and time the task record was created.		**x**
createdOnBehalfOf	**CREATED_ON_BEHALF_OF_ID**			**x**	TContact	object	When not null, it indicates that the record was created through an e-	**X**	

							billing system or other external application. The contact referenced will often be a vendor in TeamConnect.		
currentAssignee-->	**CURRENT_ASSIGNEE_ID** (**NUMBER**)	**Assigned To** (**General** block)		**x**	JTaskAssignee	object	Links to the definition information of the user who is currently assigned to the task.		**x**
currentAssigneeUser-->	**CURRENT_ASSIGNEE_USER_ID** (**NUMBER**)			**x**	YUser	object	The user who is currently assigned to the task. It is more efficient to use this field in searches and queries than to use the field "currentAssignee" to access user info.	**x**	**x**
defaultCategory-->	**DEFAULT_CATEGORY_ID** (**NUMBER**)	**Default Category** (**General** block)	**x**	**x**	YRecentlyViewedRecord	object	8 Default task category. --> Links to the definition information of the category.		**x**
detailList-->	**n/a**		**x**	**x**	ETaskDetail In Object Navigator, links to		8 The list of categories added to the current object.		

					Category list which appears in the UI only.		--> In the object model, the added categories link to the values that have been added for custom fields, according to field type. --> In Object Navigator, this attribute is enhanced. It links to a list of all categories defined for the object definition, rather than those added to the current object. From there, you can select a category and traverse to the list of custom fields that belong to it.		
documentFolder-->	**DOCUMENT_FOLDER_ID** **(NUMBER)**	**Documents** block		**x**	TDocument	object	Links to specific information about the document folder.		
dueDate **Before TeamConnect 3.3 SP2: dueOn**	**DUE_ON** **(DATE)**	**Due On** (**General** block)	**x**			date	The date when the task is due. Time-zone-independent.	**x**	**x**

estimatedHours	**ESTIMATED_HOURS** (NUMBER)	**Estimated Duration** (**General** block)	**x**			decimal	Amount of time (Hours.Minutes) needed to complete the task.		**x**
forwardedByAssignee-->	**FORWARDED_BY_ASSIGNEE_ID** (NUMBER)			**x**	JTaskAssignee	object	Links to the definition information of the user who forwarded the task to the current assignee.		
groupAccessList-->	**n/a**			**x**	ETaskGroupAccess		Links to the group security information entered through the Security block of the record.		
historyList-->	**n/a**	**History** tab	**x**	**x**	THistory		8 The list of all histories that are related to the task record. --> Links to specific history information for all of the task's history records.		
modifiedBy-->	**MODIFIED_BY_ID** (NUMBER)	**Modified By** (**Security** block)	**x**	**x**	YUser	object	8 User who most recently modified the task. --> Links to specific information		

							about the user account.		
modifiedOn	**MODIFIED_ON** **(DATE)**	**Modified On** (**Security** block)	**x**			date	Date and time the task was most recently modified.		
note-->	**NOTE_ID** **(NUMBER)**	**Notes** (**General** block)		**x**	JNote	object	Links to the table that contains the text entered into the **Notes** field of the record.		
postingStatus **Before TeamConnect 3.3 SP2: postingStatusIID**	**STATUS_ID** **(NUMBER)**	**Posting Status** (**General** block)	**x**			Enum	This enumeration is based on TAccount. Specifies whether the task was posted or not: 1 - Not Posted 2 - Posted 3 - Failed 4 - Posted For Approval	**x**	
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the task record.		
priority **Before TeamConnect 3.3 SP2: priorityIID**	**PRIORITY_IID** **(NUMBER)**	**Priority** (**General** block)	**x**			int	Task priority. 1 - Highest 2 - High 3 - Normal 4 - Low	**x**	**x**

							5 - Lowest		
project-->	**PROJECT_ID** **(NUMBER)**	**Project** (**General** block)	**x**	**x**	TProject	object	8 Project record for which this task record is created. --> Links to specific information about the project record.	**x**	**x**
rateAmount	**RATE_AMOUNT** **(NUMBER)**	**Rate** (**General** block)	**x**			decimal	Rate the user charges (per hour) to perform the task.	**x**	**x**
securityTypeIID	**SECURITY_TYPE_IID** **(NUMBER)**	**Security Type** (**Security** block)	**x**			int	Specifies whether the task is public or private. 0 - Public 2 - Private		
shortDescription	**SHORT_DESCRIPTION** **(VARCHAR2) (250)**	**Subject** (**General** block)	**x**			string (250)	Description of the task.		**x**
startDate **Before TeamConnect 3.3 SP2: startOn**	**START_ON** **(DATE)**	**Start On** (**General** block)	**x**			date	Date when the task was started. Time-zone-independent.	**x**	**x**
totalAmount	**TOTAL_AMOUNT** **(NUMBER)**	**Total** (**General** block)	**x**			decimal	Total amount of money spent to completing the task as the product of	**x**	**x**

							the rate (**rateAmount**) and amount of time spent (**actualHours**).		
transaction-->	**TRANSACTION_ID** (NUMBER)	**Transactions** block		**x**	RTransaction	object	Links to the transactions associated with the task record.		
userAccessList-->	**n/a**			**x**	ETaskUserAccess		Links to the user security information entered through the Security block of the record.		
version	**VERSION** (NUMBER)		**x**			int	Numeric instance that tells you how many times each row was modified.		
workStatus **Before TeamConnect 3.3 SP2: workStatusID**	**WORK_STATUS_IID** (NUMBER)	**Status** (**General** block)				int	The current status of the task. 1 - Not Started 2 - Started 3 - Completed		

1.1.32.2 LTaskActivityItem

LTaskActivityItem contains information about the activity items defined for task records. This information includes the name of the activity item, tree position, and display order. This table represents the **Activity Item** system lookup table.

Object: LTaskActivityItem (L_TASK_ACTIVITY_ITEM)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
childList-->	**n/a**						Do not apply. These lookup items cannot have a hierarchical structure.		
parent-->	**PARENT_ID** (NUMBER)								
displayOrder	**DISPLAY_ORDER** (NUMBER)	**Order**	**x**			int	The order in which the table items will be listed in the GUI, lowest number first.		
isActive	**IS_ACTIVE**	**activate** and **inactivate** buttons	**x**			int	Indicates whether the table entry is active (value 1) and can be used when editing records containing that entry. If it is inactive (value 0), records may contain that entry for reporting and historical purposes, but cannot be saved when editing, unless that entry is changed to some other, active table entry.		
name	**NAME** (VARCHAR2) (50)	**Item Name**	**x**			string (50)	Name of activity item.		
primaryKey	**PRIMARY_KEY**		**x**			int	The unique ID of the activity item.		

	(NUMBER)								
treePosition	**TREE_POSITION** **(VARCHAR2) (250)**	**Tree Position**	**x**			string (250)	The unique four-character alphanumeric code that is created when the activity item is defined.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the activity item was updated.		

1.1.32.3 JTaskAssignee

JTaskAssignee contains information about the assignees of a task record. This information includes all past assignees as well as the current assignee and the date assigned. This table represents the **Task Assignees** block on the **Assignees** tab of a task record.

Object: JTaskAssignee (J_TASK_ASSIGNEE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
assignedOn	**ASSIGNED_ON** **(DATE)**	**Assigned On** (**Assignees** block)	**x**			date	Date and time when the user was assigned to the task.		**x**
isInformed	**IS_ASSIGNEE_INFORMED** **(NUMBER)**						Currently not used.		**x**
owner -->	**TASK_ID** **(NUMBER)**			**x**	TTask	object	The task the user is assigned to.		**x**

primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the task assignee.		
user-->	**USER_ID (NUMBER)**		**x**	**x**	YUser	object	8 The user assigned to the task. --> Links to specific information about the user account.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the activity item was updated.		

1.1.32.4 ETaskDetail

ETaskDetail contains all categories added to task records. Each time a category is added to a task record, an entry is made in this table in the database.

In Object Navigator, a list of all categories that have been defined for the current object definition appears. This list is labeled **Category**. From each category, you can traverse to a table called **Detail Fields**, which displays all custom fields belonging to the category you selected.

For more information about how custom field values are stored in the database, see Object Model: Custom Fields.

Object: ETaskDetail (E_TASK_DETAIL)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
category-->	**CATEGORY_ID (NUMBER)**	**Category**	**x**	**x**	YRecentlyViewedRecord	object	The category of the task with which the details are associated.		
isManual	**IS_MANUAL (NUMBER)**		**x**			boolean	Specifies whether the category is added manually or		

							automatically to the task. Categories are added automatically when they are the parent of the category that is added manually. 0 - Added Automatically 1 - Added Manually		
owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**			**x**	TTask	object	The task with which the custom fields are associated.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the task category record.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the added category was updated.		

1.1.32.5 ETaskUserAccess

ETaskUserAccess contains the user access information that is set in the **Security** block of each task record. For example, it includes whether each user that is listed in the Security block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: ETaskUserAccess (E_TASK_USER_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz

isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM (NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed		

							or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(CHAR)**	**Option**	**x**			char	Indicates whether the user is allowed or denied access to the record. a - Allow d - Deny		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the user rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	TTask	object	8 The task record with which this user access setting is associated. --> Links to the specific information about the task record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the user access rights.		
user-->	**USER_ID** **(NUMBER)**	**List**	**x**	**x**	YUser	object	8 Specifies the user the access rights pertain to. --> Links to the specific information about the user account.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the user access		

							rights have been updated.		

1.1.32.6 ETaskGroupAccess

ETaskGroupAccess contains the group access information that is set in the **Security** block of each task record. For example, it includes whether each group that is listed in the Security block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: ETaskGroupAccess (E_TASK_GROUP_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial.		

							Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM** **(NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(CHAR) (1)**	**Option**	**x**			char	Indicates whether the group is allowed or denied access to the record. a - Allow d - Deny		
group-->	**GROUP_ID** **(NUMBER)**	**List**	**x**	**x**	YGroup	object	8 Specifies the group that the access rights pertain to. --> Links to the specific information about the group account.		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the group access rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually		

							1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID (NUMBER)**		**x**	**x**	TTask	object	8 The task record with which this group access setting is associated. --> Links to the specific information about the task record.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	Unique ID for the group access rights.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the group access rights have been updated.		

1.1.33 Object Model: Invoices

The tables in this appendix provide information about the object model as it relates to invoices.

1.1.33.1 TInvoice

TInvoice contains the general information for an invoice record. This information includes the invoice's creation date, received date, posting status, vendor, currency, exchange rate, and so on. This table also links to the list of custom fields created for the invoice categories, as well as the list of line items that have been added to the invoice.

Object: TInvoice (T_INVOICE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
categories-->	**n/a**		**x**	**x**	EInvcDetail In Object Navigator,		8 The list of categories added to the		

Before TeamConnect 3.3 SP2: detailList-->					links to **Category** list which appears in the UI only.		current object. --> In the object model, the added categories link to the values that have been added for custom fields, according to field type. --> In Object Navigator, this attribute is enhanced. It links to a list of all categories defined for the object definition, rather than those added to the current object. From there, you can select a category and traverse to the list of custom fields that belong to it.		
comment	**COMMENTS** **(VARCHAR2) (2053)**	**Comments** (**General** block)	**x**			string (2053)	The comments intended to appear on the invoice for the vendor.		

createdBy-->	**CREATED_BY_ID** **(NUMBER)**	**Created By** (**Security** block)	**x**	**x**	YUser	object	8 The user who created the invoice record. --> Links to specific information about the user account.	**x**	**x**
createdOn	**CREATED_ON** **(DATE)**	**Created On** (**Security** block)	**x**			date	Date the invoice was created.	**x**	**x**
createdOnBehalfOf	**CREATED_ON_BEHALF_OF_ID**			**x**	TContact	object	When not null, it indicates that the record was created through an e-billing system or other external application. The contact referenced will often be a vendor in TeamConnect.	**X**	
currency-->	**CURRENCY_ID** **(NUMBER)**	**Currency** (**General** block)		**x**	YCurrencyInfo	object	8 The currency selected for the invoice. --> Links to the definition information of the currency.		

defaultCategory-->	**DEFAULT_CATEGO RY_ID** (**NUMBER**)	**Default Category** (**Categories** block)			YRecentlyViewedRecord	object	8 The default category for the invoice. --> Links to the definition information of the category.		**x**
documentFolder-->	**DOCUMENT_FOLDE R_ID** (**NUMBER**)	**Documents** block		**x**	TDocument	object	Links to specific information about the document folder.		
exchangeRate	**EXCHANGE_RATE** (**NUMBER**)		**x**			decimal	The exchange rate of the invoice.		
expenseAdjustmentTotal	**EXPENSE_ADJUST MENT_TOTAL** (**NUMBER**)	**Invoice Header - Total Fees & Expenses** section, **Expenses** row **Adjustments** column intersection	**x**			decimal	Sum of line item expense adjustments	**x**	**x**
expenseDiscountTotal	**EXPENSE_DISCOU NT_TOTAL** (**NUMBER**)	**Invoice Header - Total Fees & Expenses** section, **Expenses** row **Discoun**	**x**			decimal	Sum of line item expense original discounts	**x**	**x**

		ts column intersection							
expenseTaxTotal	**EXPENSE_TAX_TOTAL (NUMBER)**	**Invoice Header - Total Fees & Expenses** section, **Expenses** row **Taxes** column intersection	**x**			decimal	(Taxable line item expense net total) *(Invoice tax rate)	**x**	**x**
feeAdjustmentTotal	**FEE_ADJUSTMENT_TOTAL (NUMBER)**	**Invoice Header - Total Fees & Expenses** section, **Fees** row **Adjustments** column intersection	**x**			decimal	Line item fee adjustments total.	**x**	**x**
feeDiscountTotal	**FEE_DISCOUNT_TOTAL (NUMBER)**	**Invoice Header - Total Fees & Expenses** section, **Fees** row **Discounts** column intersection	**x**			decimal	Line item fee original discounts total.	**x**	**x**

feeTaxTotal	**FEE_TAX_TOTAL (NUMBER)**	**Invoice Header - Total Fees & Expenses** section, **Fees** row **Taxes** column intersection	**x**			decimal	(Taxable line item fee net total) * (Invoice tax rate)	**x**	**x**
groupAccessList-->	**n/a**			**x**	EInvcGroupAccess		Links to the group security information entered through the Security block of the record.		
hasNonUSTax	**HAS_NON_US_TAX**		**x**			boolean	If any line items have non-US tax codes entered in them, this value is TRUE, else it is FALSE.		
historyList-->	**n/a**	**History** tab	**x**	**x**	THistory	object	8 The list of all histories that are related to the invoice record. --> Links to specific history information for all of the invoice's history records.	**x**	

invoiceAdjustmentTotal	**INVOICE_ADJUSTMENT_TOTAL (NUMBER)**	**Invoice Header - Total Fees & Expenses** section, **Line Items Total** row **Adjustments** column intersection	**x**			decimal	(Fee adjustments total) + (Expense adjustments total)	**x**	**x**
invoiceDate	**INVOICE_DATE** **(DATE)**	**Invoice Date** (**General** block)	**x**			date	The date when the invoice was issued. Time-zone-independent.	**x**	
invoiceDiscountTotal	**INVOICE_DISCOUNT_TOTAL (NUMBER)**	**Invoice Header - Total Fees & Expenses** section, **Line Items Total** row **Discounts** column intersection	**x**			decimal	(Fee discounts total) + (Expense discounts total)	**x**	**x**
invoiceTaxTotal	**INVOICE_TAX_TOT AL (NUMBER)**	**Invoice Header - Total Fees & Expenses** section,	**x**			decimal	(Fee taxes total) + (Expense taxes total)	**x**	**x**

		Line Items Total row **Taxes** column intersection							
isElectronic	**IS_ELECTRONIC**		**x**			boolean	If TRUE, indicates that the invoice originated from an e-billing application such as Collaborati.		
isPercentageDiscount	**IS_PERCENTAGE_DISCOUNT**		**x**			boolean	An invoice discount in percent form.		
lineItems--> Before TeamConnect 3.3 SP2: lineItemList -->	**n/a**		**x**	**x**	JInvcLineItem		8 The list of all line items that are added to the invoice record. --> Links to the join table that contains information for all line items added to the invoice.	**x**	**x**
modifiedBy-->	**MODIFIED_BY_ID (NUMBER)**	**Modified By** (**Security** block)	**x**	**x**	YUser	object	8 The user who last modified the invoice record. --> Links to specific		

							information about the user account.		
modifiedOn	**MODIFIED_ON** **(DATE)**	**Modified On** (**Security** block)	**x**			date	The date the invoice was last modified.		
netExpenseTotal	**NET_EXPENSE_TOTAL (NUMBER)**	**Invoice Header - Total Fees & Expenses** section, **Expenses** row **Total** column intersection	**x**			decimal	(Line item expenses total) + (Expense taxes total)	**x**	**x**
netFeeTotal	**NET_FEE_TOTAL (NUMBER)**	**Invoice Header - Total Fees & Expenses** section, **Fees** row **Total** column intersection	**x**			decimal	(Line item fees net total) + (Fees taxes total)	**x**	**x**
netInvoiceTotal	**NET_INVOICE_TOT AL (NUMBER)**	**Invoice Header - Total Fees & Expenses** section, **Line Items**	**x**			decimal	(Net fees total) + (Net expenses total)	**x**	**x**

		Total row, **Total** column intersection							
nextLineItemNumber	**NEXT_LINE_ITEM_NUMBER**		**x**			int	Displays the next available line item number for this invoice.		
note-->	**NOTE_ID** (**NUMBER**)	**Notes** (**General** block)		**x**	JNote	object	Links to the table that contains the text entered into the **Notes** field of the record.		
numberString	**NUMBER_STRING** (**VARCHAR2**) (**50**)	**Number** (**General** block)	**x**			string (50)	Any alphanumeric text that describes the invoice record.		
originalExpenseTotal	**ORIGINAL_EXPENSE_TOTAL** (**NUMBER**)	**Invoice Header - Total Fees & Expenses** section, **Expenses** row **Original** column intersection	**x**			decimal	(Line item expenses Original total) + (Line item expenses Original discounts total)	**x**	**x**
originalFeeTotal	**ORIGINAL_FEE_TOTAL**	**Invoice Header - Total Fees &**	**x**			decimal	(Line item fees Original total) + (Line item	**x**	**x**

	(NUMBER)	**Expenses** section, **Fees** row **Original** column intersection					fees Original discounts total)		
originalInvoiceTotal	**ORIGINAL_INVOICE_TOTAL (NUMBER)**	**Invoice Header - Total Fees & Expenses** section, **Line Items Total** row **Original** column intersection	**x**			decimal	(Original fees total) + (Original expenses total)	**x**	**x**
periodEndDate **Before TeamConnect 3.3 SP2: periodEndOn**	**PERIOD_END_ON** **(DATE)**	**Period To** (**General** block)	**x**			date	The end date of the invoice period. Time-zone-independent.		
periodStartDate **Before TeamConnect 3.3 SP2: periodStartOn**	**PERIOD_START_ON** **(DATE)**	**Period From** (**General** block)	**x**			date	The start date of the invoice period. Time-zone-independent.		
statusType **Before TeamConnect 3.3 SP2:**	**STATUS_TYPE_IID** **(NUMBER)**	**Status** (**General** block)	**x**			int	The posting status of the invoice.	**x**	

postingStatusIID							1 - Not Submitted 2 - Posted 3 - Not Posted 5- Rejected*		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the invoice record.		
receivedDate	**RECEIVED_DATE** **(DATE)**	**Date Received** (**General** block)	**x**			date	The date when the invoice was received. Time-zone-independent.	**x**	
referenceNumber	**REFERENCE_NUMB ER** **(VARCHAR 50)**	**Reference Number**				string	If this invoice is of type Credit Note, then referenceNumber matches the numberString value of the original standard invoice for which this credit is intended.		
securityTypeIID	**SECURITY_TYPE_IID** **(NUMBER)**	**Security Type** (**Security** block)	**x**			int	The security type of the invoice record . 0 - Public 2 - Private		

submittedElectronically **Before TeamConnect 3.3 SP2: isElectronic**	**IS_ELECTRONIC**		**x**			boolean		**x**	
submittedTotal **Before TeamConnect 3.3 SP2: manualTotal**	**MANUAL_AMOUNT** **(NUMBER)**	**Manual Amount** (**General** block)	**x**			decimal	The total amount of the invoice as entered manually by the user.	**x**	
taxRate	**TAX_RATE** **(NUMBER)**						Currently not used in TeamConnect.		
transaction->	**TRANSACTION_ID** **(NUMBER)**	**Transaction** block		**x**	RTransaction	object	Contains the last transaction associated with the invoice that was either posted to or voided from an account.		
typeIID	**TYPE_IID** **(NUMBER)**	**Invoice Type** (**General** block)	**x**			Enum	The general type of the invoice. 0 - Standard (STANDARD) 1 - Accrual (ACCRUAL) 2 -Credit Note (CREDIT_NOTE)	**x**	

							3 - Shadow Invoice (SHADOW)		
userAccessList-->	**n/a**			**x**	EInvcUserAccess		Links to the user security information entered through the Security block of the record.		
vendor-->	**VENDOR_ID** (**NUMBER**)	**Vendor** (**General** block)	**x**	**x**	TContact	object	8 The contact record of the vendor associated with the invoice. --> Links to the specific information about the contact object.		
version	**VERSION** (**NUMBER**)		**x**			int	Indicates how many times the invoice record was updated.		
warnings	**WARNINGS**	**Warnings**				string (2000)	Warning message for the invoice captured by TeamConnect that is generated by another application that processes invoices.	**X**	

1.1.33.2 JInvcCategoryAdjustment

JInvcCategoryAdjustment contains information about the adjustment of line items in an invoice record. This information includes the nature of the adjustment calculation, the reason for the adjustment, and comments about the adjustment.

Object: JInvcCategoryAdjustment (J_INVC_CATEGORY_ADJUSTMENT)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
	ADJUSTED_BY_ID (NUMBER)		**x**	**x**	YUser	Number	8 The user who made the adjustment. --> Links to specific information about the user account.	**x**	**x**
adjustmentDate **Before TeamConnect 3.3 SP2: adjustedOn**	**ADJUSTED_ON (DATE)**					date	Defaults to current date/time.		
adjustmentAmount	**ADJUSTMENT_AMOUNT** **(NUMBER)**		**x**			decimal	Adjustment amount.		
adjustmentMethod **Before TeamConnect 3.3 SP2: adjustmentTypeIID**	**IS_TAXABLE** **(NUMBER)**		**x**			int	Adjustment type. One of these choices: 1 = By Amount 2 = By Percentage 3 = To Amount		
adjustmentReason-->	**ADJUSTMENT_REASON**		**x**	**x**		object	8 The selected reason for the adjustment.		

	(NUMBER)								
dcategory-->	**CATEGORY_ID** **(NUMBER)**	**Category**				object	8 The category fto which this adjustment applies.		**x**
commentsToVendor	**COMMENT_TO_VENDOR** **(VARCHAR 250))**		**x**			string (250)	Comments about the adjustment that will be visible to vendors outside your company.		
inHouseComments	**IN_HOUSE_COMMENT<** **(VARCHAR 250)**		**x**			string (250)	Comments that are private, not transmitted to vendors or other outside sources.		
iowner-->	**INVOICE_ID** **(NUMBER)**			**x**	TInvoice	object	8 The invoice to which the line item belongs. --> Links to the specific information about the invoice object.		
project-->	**PROJECT_ID** **(NUMBER)**	**Project**	**x**	**x**	TProject	object	8 The project associated with the adjustment. --> Links to the specific information about the project object.	**x**	**x**
previousNetTotal	**PREVIOUS_NET_TOTAL** **(NUMBER)**	**Original Discount**	**x**			decimal	The net total of the entire invoice immediately before the current adjustment was applied.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of this object.		

version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the object was updated.		

1.1.33.3 JInvcHeaderAdjustment

JInvcHeaderAdjustment contains information about the adjustment of line items in an invoice record. This information includes the nature of the adjustment calculation, the reason for the adjustment, and comments about the adjustment.

Object: JInvcHeaderAdjustment (J_INVC_ADJUSTMENT)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
	ADJUSTED_BY_ID (NUMBER)		**x**	**x**	YUser	Number	8 The user who made the adjustment. --> Links to specific information about the user account.	**x**	**x**
adjustmentDate **Before TeamConnect 3.3 SP2:** **adjustedOn**	**ADJUSTED_ON (DATE)**					date	Defaults to current date/ time.		
adjustmentMethod **Before TeamConnect 3.3 SP2:** **adjustmentTypeID**	**IS_TAXABLE (NUMBER)**		**x**			int	Adjustment type. One of these choices: 1 = By Amount 2 = By Percentage 3 = To Amount		

adjustmentReason-->	**ADJUSTMENT_REASON** **(NUMBER)**		**x**	**x**		object	8 The selected reason for the adjustment.		
adjustmentTargetIID **Before TeamConnect 3.3 SP2: adjustmentTarget**	**ADJUSTMENT_TARGET_ID** **(NUMBER)**		**x**			int	The target area to be impacted by the adjustment. One of these choices: 1 = Total Fees 2 = Total Expenses 3 = Total Invoice		
adjustmentTotalExpenses	**ADJUSTMENT_TOTAL_EXPENSES** **(NUMBER)**		**x**			decimal	Adjustment amount to be applied only to the expense portion of the invoice. Calculation is done as specified by attribute adjustmentTypeIID.		
adjustmentTotalFees	**ADJUSTMENT_TOTAL_FEES** **(NUMBER)**		**x**			decimal	Adjustment to be applied only to the fee portion of the invoice. Calculation is done as specified by attribute adjustmentTypeIID.		
adjustmentTotalInvoice	**ADJUSTMENT_TOTAL_INVOICES** **(NUMBER)**		**x**			decimal	Adjustment to be applied to the entire amount of the invoice.		

							Calculation is done as specified by attribute adjustmentTypeIID.		
commentsToVendor	**COMMENT_TO_VENDOR** **(VARCHAR 250))**		**x**			string (250)	Comments about the adjustment that will be visible to vendors outside your company.		
inHouseComments	**IN_HOUSE_COMMENT<** **(VARCHAR 250)**		**x**			string (250)	Comments that are private, not transmitted to vendors or other outside sources.		
owner--> **Before TeamConnect 3.3 SP2: invoice-->**	**INVOICE_ID** **(NUMBER)**			**x**	TInvoice	object	8 The invoice to which the line item belongs. --> Links to the specific information about the invoice object.		
previousNetTotal	**PREVIOUS_NET_TOTAL** **(NUMBER)**	**Original Discount**	**x**			decimal	The net total of the entire invoice immediately before the current adjustment was applied.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of this object.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the object was updated.		

1.1.33.4 JInvcLineItem

JInvcLineItem contains information about the line items in an invoice record. This information includes the project to which the line item is associated, amount, line item type (such as task or expense), activity, task category, timekeeper, and service date. This table represents the **Line Items** block on the **Line Items** tab of an invoice record.

Object: JInvcLineItem (J_INVC_LINE_ITEM)

Attribute	Database column name	Field in user interface	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
activity **Before TeamConnect 3.3 SP2: activityItem**	**ACTIVITY_ITEM_ID** **(NUMBER)**	**Activity Item**	**x**	**x**	LTaskActivityItem	object	8 The activity item associated with the line item of type task. --> Links to the definition information for the task activity item.	**x**	**x**
adjustedDiscount **Before TeamConnect 3.3 SP2: netDiscount**	**ADJUSTED_DISCOUNT** **(NUMBER)**	**Net Discount**	**x**			decimal	The net discount of the line item.		
adjustedQuantity **Before TeamConnect 3.3 SP2: netQuantity**	**ADJUSTED_QUANTITY** **(NUMBER)**	**Net Quantity**	**x**			decimal	The net quantity of the line item.		
adjustedRate	**ADJUSTED_RATE**	**Net Rate**	**x**			decima	The net rate of the line		

Before TeamConnect 3.3 SP2: netRate	**(NUMBER)**					l	item.		
adjustedTotal **Before TeamConnect 3.3 SP2: netTotal**	**TOTAL_AMOUNT** **(NUMBER)**	**Net Total**	**x**			decimal	The net total of the line item.		
adjustments **Before TeamConnect 3.3 SP2: adjustmentList**			**x**	**x**		list	8 Any invoice adjustments that have been entered against this line item. --> Links to any adjustments that have been entered against this line item.		
categories--> **Before TeamConnect 3.3 SP2: detailList-->**			**x**	**x**	ELitmDetail	list	8 The custom field detail values that have been entered against this line item. --> Links to the collection of custom field detail values that have been entered against this line item.		

createdBy-->	**CREATED_BY_ID** **(NUMBER)**	**Created By** (**Security** block)	**x**	**x**	YUser	object	8 The user who created the invoice record. --> Links to specific information about the user account.	**x**	**x**
createdOn	**CREATED_ON** **(DATE)**	**Created On** (**Security** block)	**x**			date	Date the invoice was created.	**x**	**x**
currencyItem-->	**CURRENCY_ITEM_ID** **(NUMBER)**						Currently not used.		
defaultCategory	**DEFAULT_CATEGORY_ID**	**Default Category** (**Categories** block)	**x**	**x**	YRecentlyViewedRecord	object	8 The default category for the line item. --> Links to the definition information of the category.		
defaultCurrencyNetDiscount	**DEFAULT_CURR_DISCOUNT** **(NUMBER)**		**x**			decimal	The net discount of the line item in the system's default currency.		
defaultCurrencyNetRate	**DEFAULT_CURR_RATE**		**x**			decimal	The net rate of the line item in the system's		

	(NUMBER)						default currency.		
defaultCurrencyNetTotal	**DEFAULT_CURR_TOTAL** **(NUMBER)**		**x**			decimal	The net total of the line item in the system's default currency.		
expenseCategory-->	**EXPENSE_TYPE_ID** **(NUMBER)**	**Category** (when **Expense** is selected in the **Task/Expense** field)	**x**	**x**	YRecentlyViewedRecord	object	8 The expense category of the line item, if the line item is an expense. --> Links to the definition information of the category.	**x**	**x**
groupAccessList-->				**x**	ELitmGroupAccess	list	8 The non-default group access rights that have been granted against this line item. --> Links to the collection of non-default group access rights that have been granted against this line item.		
hasWarnings	**HAS_WARNINGS**		**x**			boolea	Indicates whether		

	(NUMBER)					n	warning conditions have been noted for this line item.		
involved-->	**INVOLVED_ID** **(NUMBER)**		**x**	**x**			Currently not used.		
itemNumber **Before TeamConnect 3.3 SP2: itemNum**	**ITEM_NUMBER**		**x**			int	The number assigned to the line item at the time it was created.Invoice item numbers begin at 1 and increment as high as necessary to accommodate the invoice's line item.If a line item is deleted, its item number is not reused for any existing or future line item.		
modifiedBy-->	**MODIFIED_BY_ID** **(NUMBER)**		**x**	**x**	YUser	object	8 The user who last modified the line item. --> Links to specific information about the		

							user account.		
modifiedOn	**MODIFIED_ON (DATE)**	**Last Modified On**	**x**			date	The date that the line item was modified.		
note Before TeamConnect 3.3 SP2: shortDescription	**ITEM_DESCRIPTION (VARCHAR2) (250)**	**Description**	**x**			string (2000)	The description of the line item.		**x**
originalDiscount	**DISCOUNT_PERCENT (NUMBER)**	**Original Discount**	**x**			decimal	The original discount of the line item.		
originalQuantity	**QUANTITY (NUMBER)**	**Original Quantity**	**x**			decimal	The original quantity of the line item.		
originalRate	**UNIT_PRICE (NUMBER)**	**Original Rate**	**x**			decimal	The original rate of the line item.		
originalTotal	**GROSS_AMOUNT (NUMBER)**	**Original Total**	**x**			decimal	The original total of the line item.		
owner-->	**INVOICE_ID (NUMBER)**			**x**	TInvoice	object	8 The invoice to which the line item belongs. --> Links to the specific information about the invoice object.		

primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the line item.		
project-->	**PROJECT_ID** **(NUMBER)**	**Project**	**x**	**x**	TProject	object	8 The project associated with the line item. --> Links to the specific information about the project object.	**x**	**x**
securityTypeID	**SECURITY_TYPE_ID** **(NUMBER)**		**x**			int	Security type. One of these values: 0= Public 2 = Private		
serviceDate	**SERVICE_DATE** **(DATE)**	**Service Date**	**x**			date	The date when the service to which the line item refers was rendered. Time-zone-independent.		
taskCategory-->	**TASK_CATEGORY_ID** **(NUMBER)**	**Category** (when **Task** is selected in the **Task/Expense** field)	**x**	**x**	YRecentlyViewedRecord	object	8 The task category of the line item, if the line item is a task. --> Links to the definition information of the category.	**x**	**x**

taxable **Before TeamConnect 3.3 SP2: isTaxable**	**IS_TAXABLE** **(NUMBER)**		**x**				Currently not used.		
timekeeper **Before TeamConnect 3.3 SP2: vendorRep-->**	**VENDOR_ID** **(NUMBER)**	**Timekeeper**	**x**	**x**	TContact	object	8 The contact who performed the service or provided the goods. --> Links to the specific information about the contact object.	**x**	**x**
totalAdjustment	**TOTAL_** **ADJUSTMENT**	**Total Adjustment**	**x**			decimal	The total amount of all adjustments made to the line item.	**x**	
type **Before TeamConnect 3.3 SP2: typeIID**	**TYPE_IID** **(NUMBER)**	**Task/ Expense**	**x**			int	The type of the line item. E - Expense T - Task	**x**	**x**
userAccessList-->				**x**	ELitmUser Access	list	8 The non-default user access rights that have been granted against this line item. --> Links to the collection of non-default user access		

							rights that have been granted against this line item.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the invoice record was updated.		
warnings	**LINEITEM_WARNINGS (NUMBER)**		**x**			string (2000)	Collected text of warning messages related to this line item.		

1.1.33.5 JInvcNonUSTax

JInvcNonUSTax contains information about the non-U.S. tax amounts in line items in an invoice record.

Object: JInvcAdjustment (J_INVC_NON_US_TAX)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of this object.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the object was updated.		
owner-->	**INVOICE_ID (NUMBER)**			**x**	TInvoice	object	8 The invoice to which the tax		

Before TeamConnect 3.3 SP2: invoice-->							amount belongs. --> Links to the specific information about the invoice object.		
project-->	**PROJECT_ID** **(NUMBER)**	**Project**	**x**	**x**	TProject	object	8 The project associated with the tax amount. --> Links to the specific information about the project object.	**x**	**x**
nonUSTaxType-->	**NON_US_TAX_TYPE** **(NUMBER)**	**Non-US Tax Type**		**x**	LInvcNonUSTaxType	object	8 The tax type category associated with the tax amount.	**x**	**x**
taxAmount	**TAX_AMOUNT** **(NUMBER)**	**Tax Amount**	**x**			decimal	The tax amount in original, local currency.		
defaultCurrencyTaxAmount	**DEFAULT_CURR_TAX_AMOUNT** **(NUMBER)**	**Tax Amount**	**x**			decimal	The tax amount in the invoice's default currency.		

1.1.33.5.1 LInvcNonUSTaxType

LInvcNonUSTaxType contains information about the tax types available when creating invoice line items containing non-U.S. taxes. This information includes the description of the tax type, tree position, and display order. This table represents the **Non-US Tax Type** tab in the **Invoice** object definition in Designer.

Object: LInvcNonUSTaxType (L_INVC_NON_US_TAX_TYPE)

Attribute	Database column name	Field in UI	End of	Bridge	Links to object table:	Data type	Comments	Commonly used in:

			path					Rules	
displayOrder	DISPLAY_ORDER (NUMBER)	Order	x			int	The order in which the items will be listed in the GUI, lowest number first.		
name	NAME (VARCHAR2) (50)	Item Name	x			string (50)	The description of the tax type.		
parent -->	PARENT_ID (NUMBER)	Show items in node	x	x	LInvcNonUSTaxType	object	8 The parent item of the current item. --> Links to definition information for the parent tax type.		
primaryKey	PRIMARY_KEY (NUMBER)		x			int	The unique ID of the tax type list item.		
treePosition	TREE_POSITION (VARCHAR2) (250)	Tree Position	x			string (250)	The unique four-character alphanumeric code that is created when the tax type is defined.		
version	VERSION (NUMBER)		x			int	Indicates how many times the skill type list item was updated.		
uniqueCode	UNIQUE_CODE (VARCHAR2) (250)	Tax Code	x			string (20)	The unique 20-character alphanumeric code that the solution developer specifies when entering a new tax code.		

isActive	**IS_ACTIVE** **(BOOLEAN)**		**x**			boolean	If TRUE, the tax type is active and can be selected by the end user.		

1.1.33.6 JInvcTimekeeperAdjustment

JInvcTimekeeperAdjustment contains information about the adjustment of line items in an invoice record. This information includes the nature of the adjustment calculation, the reason for the adjustment, and comments about the adjustment.

Object: JInvcTimekeeperAdjustment (J_INVC_TIMEKEEPER_ADJUSTMENT)

Attribute	Database column name	Field in user interface	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	**Temp/Wiz**
	ADJUSTED_BY_ID (NUMBER)		**x**	**x**	YUser	Number	8 The user who made the adjustment. --> Links to specific information about the user account.	**x**	**x**
adjustmentDate **Before TeamConnect 3.3 SP2: adjustedOn**	**ADJUSTED_ON (DATE)**					date	Defaults to current date/time.		
adjustmentMethod **Before TeamConnect 3.3 SP2: adjustmentTypeIID**	**IS_TAXABLE** **(NUMBER)**		**x**			int	Adjustment type. One of these choices: 1 = By Amount 2 = By Percentage 3 = To Amount		

adjustmentReason-->	**ADJUSTMENT_REASON (NUMBER)**		**x**	**x**		object	8 The selected reason for the adjustment.		
commentsToVendor	**COMMENT_TO_VENDOR (VARCHAR 250))**		**x**			string (250)	Comments about the adjustment that will be visible to vendors outside your company.		
inHouseComments	**IN_HOUSE_COMMENT< (VARCHAR 250)**		**x**			string (250)	Comments that are private, not transmitted to vendors or other outside sources.		
owner-->	**INVOICE_ID (NUMBER)**			**x**	TInvoice	object	8 The invoice to which the line item belongs. --> Links to the specific information about the invoice object.		
previousNetTotal	**PREVIOUS_NET_TOTAL (NUMBER)**	**Original Discount**	**x**			decimal	The net total of the entire invoice immediately before the current adjustment was applied.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of this object.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the object was updated.		

1.1.33.7 JLitmAdjustment

JLitmAdjustment contains information about the adjustment of line items in an invoice record. This information includes the nature of the adjustment calculation, the reason for the adjustment, and comments about the adjustment.

Object: JILitmAdjustment (J_LITM_ADJUSTMENT)

Attribute	Database column name	Field in user interface	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp / Wiz
	ADJUSTED_BY_ID (NUMBER)		**x**	**x**	YUser	Number	8 The user who made the adjustment. --> Links to specific information about the user account.	**x**	**x**
adjustmentDate **Before TeamConnect 3.3 SP2: adjustedOn**	**ADJUSTED_ON (DATE)**					date	Defaults to current date/time.		
adjustmentMethod **Before TeamConnect 3.3 SP2: adjustmentTypeIID**	**ADJUSTMENT_TYPE_ID** **(NUMBER)**		**x**			int	Adjustment type. One of these choices: 1 = By Amount 2 = By Percentage 3 = To Amount		
adjustmentReason-->	**ADJUSTMENT_REASON** **(NUMBER)**		**x**	**x**		object	8 The selected reason for the adjustment.		
adjustmentTarget **Before TeamConnect 3.3**	**ADJUSTMENT_TARGET_ID** **(NUMBER)**		**x**			int	The target area to be impacted by the adjustment. One of these choices:		

SP2: adjustmentTargetIID							1 = Total Fees 2 = Total Expenses 3 = Total Invoice		
adjustmentQuantity	**ADJUSTMENT_QUANTITY** **(NUMBER)**		**x**			decimal	Adjustment quantity to be applied to the existing line item quantity.		
adjustmentAmount	**ADJUSTMENT_AMOUNTS** **(NUMBER)**		**x**			decimal	Adjustment amount to be applied to the line item cost. Calculation is done as specified by attribute adjustmentTypeIID.		
commentsToVendor	**COMMENT_TO_VENDOR** **(VARCHAR 2000)**		**x**			string (2000)	Comments about the adjustment that will be visible to vendors outside your company.		
inHouseComments	**IN_HOUSE_COMMENT<** **(VARCHAR 250)**		**x**			string (250)	Comments that are private, not transmitted to vendors or other outside sources.		
LineItemID	**LINEITEM_ID** **(NUMBER)**			**x**	JInvcLineItem	object	8 The invoice line item to which the adjustment belongs. --> Links to the specific information about the line item.		
previousNetTotal	**PREVIOUS_NET_TOTAL** **(NUMBER)**	**Original Discount**	**x**			decimal	The net total of the entire invoice immediately before the current		

							adjustment was applied.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of this object.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the object was updated.		

1.1.33.8 EInvcDetail

EInvcDetail contains all categories added to invoice records. Each time a category is added to an invoice record, an entry is made in this table in the database.

In Object Navigator, **EInvcDetail** is not displayed when you traverse using the **detailList-->** bridge. Instead, a list of all categories that have been defined for the object definition appears. This list is labeled **Category**. From each category, you can traverse to a table called **Detail Fields**, which displays all custom fields belonging to the category you selected.

For more information about how custom field values are stored in the database, see Object Model: Custom Fields.

Object: EInvcDetail (E_INVC_DETAIL)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
category-->	**CATEGORY_ID (NUMBER)**	**Category**	**x**	**x**	YRecentlyViewedRecord	object	The category of the invoice with which the details are associated.		
owner-->	**ENTERPRISE_OBJECT_ID (NUMBER)**			**x**	TInvoice	object	The invoice with which the custom fields are associated.		

isManual	**IS_MANUAL (NUMBER)**		**x**			boolean	Specifies whether the category is added manually or automatically to the invoice. Categories are added automatically when they are the parent of the category that is added manually. 0 - Added Automatically 1 - Added Manually		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the invoice category record.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the added category was updated.		

1.1.33.8.1 ELitmDetail

ELitmDetail organizes the detail values for an invoice line item (JInvcLineItem object). Detail values originate from custom fields associated with the category assigned to the line item.

There can be multiple ELitmDetail objects for each JInvcLineItem. There is one ELitmDetail for each category used in JInvcLineItem, including any parent categories, plus the Root category. For example, if the category assigned to the JInvcLineItem object were **Root : Expense : Parking**, there would be three associated ELitmDetail objects: one for Root, one for Expense, and one for Parking.

Note that in ELitmDetail there are attributes for multiple lists of detail values. However, in any given ELitmDetail object, it is possible that only some of those attributes will really have values. Since an ELitmDetail object has a specific category, only the custom fields that match the category will be used. Thus there will only be values for the field types that those custom fields represent.

Object: ELitmDetail (E_LITM_DETAIL)

Attribute	Database column name	End of	Bridge	Links to object table:	Data typ	Comments	Commonly used in:

							Rules	Temp/Wiz
isManual	**IS_MANUAL (NUMBER)**	**x**			boolean	Specifies whether the category is added manually or automatically to the line item. Categories are added automatically when they are the parent of the category that is added manually. 0 - Added Automatically 1 - Added Manually		
primaryKey	**PRIMARY_KEY (NUMBER)**	**x**			int	Unique ID automatically assigned to this custom field value.		
version	**VERSION (NUMBER)**	**x**			int	Indicates how many times the line item detail was updated.		
category-->	**CATEGORY_ID (NUMBER)**		**x**	YRecentlyViewedRecord	object	8 The category to which this detail belongs. Links to a category to which the line item belongs. Note that most line items belong to multiple categories - a specific category, plus any parent category, grandparent category, etc.		
owner-->	**ENTERPRISE_OBJECT_ID (NUMBER)**	**x**	**x**	JInvcLineItem	object	8 The invoice line item record to which this detail belongs.		

						--> Links to information about the invoice record to which the line item belongs.		

1.1.33.9 ELitmGroupAccess

ELitmGroupAccess controls the privileges that a user group has for a specific invoice line item.

Object: ELitmGroupAccess (E_LITM_GROUP_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
accessIID	**OPERATION_IID** **(NUMBER)**	**Read**, **Update**, **Delete**, or **Perm** check boxes	**x**			int	Specifies which of the following operations have been selected: 64 - Read 2 - Update 4 - Delete 8 - Set Permission This attribute contains the sum of all operations that the group is allowed or denied access to. For example, if the group is denied access to **Delete** and **Perm**, this column would contain the value 12 [=4 (Delete) + 8 (Set Perm)] This attribute does not determine whether the operations are allowed or denied.		

allowDenyIID	**ALLOW_DENY_IID** **(NUMBER)**	**Option**	**x**			char	Indicates whether the group is allowed or denied access to the record. a - Allow d - Deny		
group-->	**GROUP_ID** **(NUMBER)**	**List**	**x**	**x**	YGroup	object	8 Specifies the group that the access rights pertain to. --> Links to the specific information about the group account.		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the group access rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	JInvcLineItem	object	8 The invoice line item record with which this group access setting is associated. --> Links to the specific information about the invoice line item record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the group access rights.		

version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the group access rights have been updated.		

1.1.33.10 ELitmUserAccess

ELitmUserAccess controls the privileges that a user has for a specific invoice line item.

Object: ELitmUserAccess (E_LITM_USER_ACCESS)

Attribute	Database column name	Field in user interface	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		

isDelete	**IS_DELETE** **(NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM** **(NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(NUMBER)**	**Option**	**x**			char	Indicates whether the group is allowed or denied access to the record. a - Allow d - Deny		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the group access rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually		

							1 - Assigned automatically		
owner->	**ENTERPRISE_OBJECT_ID (NUMBER)**		**x**	**x**	JInvcLineItem	object	8 The invoice line item record with which this group access setting is associated. --> Links to the specific information about the invoice line item record.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	Unique ID for the group access rights.		
user	**USER_ID (NUMBER)**	**List**	**x**	**x**	YUser	object	8 Specifies the user that the access rights pertain to. --> Links to the specific information about the user account.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the group access rights have been updated.		

1.1.33.11 EInvcUserAccess

EInvcUserAccess contains the user access information that is set in the **Security** block of each invoice record. For example, it includes whether each user that is listed in the Security block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EInvcUserAccess (E_INVC_USER_ACCESS)

Attribute	Database column name	Field in user interface	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz

isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM (NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are		

							allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(NUMBER)**	**Option**	**x**			char	Indicates whether the user is allowed or denied access to the record. a - Allow d - Deny		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the user rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	TInvoice	object	8 The invoice record with which this user access setting is associated. --> Links to the specific information about the invoice record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the user access rights.		
user-->	**USER_ID** **(NUMBER)**	**List**	**x**	**x**	YUser	object	8 Specifies the user the access rights pertain to. --> Links to the specific information about the user account.		

version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the user access rights have been updated.		

1.1.33.12 EInvcGroupAccess

EInvcGroupAccess contains the group access information that is set in the **Security** block of each invoice record.

For example, it includes whether each group that is listed in the **Security** block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EInvcGroupAccess (E_INVC_GROUP_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ (NUMBER)**	**Read** checkbox	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isManual	**IS_MANUAL (NUMBER)**		**x**			boolean	Indicates whether the group access rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner -->	**ENTERPRISE_OBJECT_ID**		**x**	**x**	TInvoice	object	8 The invoice record with which this group access		

	(NUMBER)						setting is associated. --> Links to the specific information about the invoice record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the group access rights.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the group access rights have been updated.		

1.1.33.13 YCurrencyInfo

YCurrencyInfo contains information about currencies defined in the multi-currency lookup table, which are used to display currency in financial fields, including the currency name, symbol, code, and exchange rate.

Object: YCurrencyInfo (Y_CURRENCY_INFO)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
code	**CODE** **(VARCHAR) (3)**	**Currency code**	**x**			string (3)	The universal 3-digit code of the currency item.		
createdBy-->	**CREATED_BY_ID** **(NUMBER)**		**x**	**x**	YUser	object	8 The user who created the currency item. --> Links to specific information about the user account.		
createdOn	**CREATED_ON** **(DATE)**		**x**			date	The date that the currency item was created.		

modifiedBy-->	**MODIFIED_BY_ID (NUMBER)**		**x**	**x**	YUser	object	8 The user who last modified the currency item. --> Links to specific information about the user account.		
modifiedOn	**MODIFIED_ON (DATE)**	**Last Modified On**	**x**			date	The date that the currency item was modified.		
name	**NAME (VARCHAR2) (50)**	**Name**	**x**			string (50)	The name of the currency item.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the currency item.		
rate	**RATE (NUMBER)**	**Exchange Rate**	**x**			decimal	The exchange rate of the currency item.		
symbol	**SYMBOL (VARCHAR) (3)**	**Symbol**	**x**			string (3)	The ISO symbol of the currency item.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the currency item was updated.		

JInvcHeaderAdjustment contains information about the adjustment of line items in an invoice record. This information includes the nature of the adjustment calculation, the reason for the adjustment, and comments about the adjustment.

1.1.34 Object Model: Histories

The tables in this appendix provide information about the object model as it relates to histories.

1.1.34.1 THistory

THistory contains the general information for a history record. This information includes the user who created the history, creation date, short description, history text, default category and so on. This table links to the list of custom fields created for the history categories and related information contained in other tables.

Object: THistory (T_HISTORY)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
archivedOn	**ARCHIVED_ON (NUMBER)**	**Date Time** (**General** block)	**x**			date Time	Date and time when the history was created, as entered by the user.	**x**	
categories--> Before TeamConnect 3.3 SP2: detailList -->	**n/a**		**x**	**x**	EHistDetail In Object Navigator, links to **Category** list which appears in the UI only.		8 The list of categories added to the current object. --> In the object model, the added categories link to the values that have been added for custom fields, according to field type. --> In Object Navigator, this attribute is enhanced. It links to a list of all categories		

							defined for the object definition, rather than those added to the current object. From there, you can select a category and traverse to the list of custom fields that belong to it.		
createdBy-->	**CREATED_BY_ID** **(NUMBER)**	**Created By** (**Security** block)	**x**	**x**	YUser	object	8 User who created the history record. --> Links to specific information about the user account.	**x**	**x**
createdOn	**CREATED_ON** **(DATE)**	**Created On** (**Security** block)	**x**			date	Date and time the history record was created.	**x**	**x**
createdOnBehalfOf	**CREATED_ON_BEHALF_OF_ID**			**x**	TContact	object	When not null, it indicates that the record was created through an e-billing system or other external application. The contact referenced will often be a vendor in	**X**	

							TeamConnect .		
defaultCategory-->	**DEFAULT_CATEGORY_ID (NUMBER)**	**Default category** (**General** block)	**x**	**x**	YRecentlyViewedRecord	object	The default category of the history record.	**x**	**x**
documentFolder-->	**DOCUMENT_FOLDER_ID (NUMBER)**			**x**	TDocument	object	Links to specific information about the document folder.		
enteredBy-->	**ENTERED_BY_ID (NUMBER)**	**Entered by** (**General** block)	**x**	**x**	YUser	object	8 User who created the history record. ***Note:*** *This is the same user as in the* ***createdBy*** --> Links to specific information about the user account.	**x**	**x**
groupAccessList-->	**n/a**			**x**	EHistGroupAccess		Links to the group security information entered through the Security block of the record.		

modifiedBy-->	**MODIFIED_BY_ID** **(NUMBER)**	**Modified By** (**Security** block)	**x**	**x**	YUser	object	8 User who last modified the record. --> Links to specific information about the user account.		
modifiedOn	**MODIFIED_ON** **(DATE)**	**Modified On** (**Security** block)	**x**			date	Date and time the record was last modified.		
objectID	**OBJECT_ID** **(NUMBER)**	displayed as the Parent Object (**General** block)	**x**			int	The object record to which the history record belongs.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the history record.		
securityTypeIID	**SECURITY_TYPE_IID** **(NUMBER)**	**Security Type** (**Security** block)	**x**			int	Specifies whether the history record is public or private. 0 - Public 2 - Private		
shortDescription	**SHORT_DESCRIPTION** **(VARCHAR2) (250)**		**x**			string (250)	The first 250 characters in the Description field.		**x**

text	**TEXT** (VARCHAR)	**Description** (**General** block)	**x**			long String	The contents of the Description field.		**x**
uniqueCode	**PARENT_UNIQUE_CODE** (VARCHAR2) (4)		**x**			string (4)	The four-character alphanumeric code that uniquely identifies the object to which the history record belongs.		
userAccessList-->	**n/a**			**x**	EHistUserAccess		Links to the user security information entered through the Security block of the record.		
version	**VERSION** (NUMBER)		**x**			int	Indicates the number of times the history record has been updated.		

1.1.34.2 EHistDetail

EHistDetail contains all categories added to history records. Each time a category is added to a history record, an entry is made in this table in the database.

In Object Navigator, **EHistDetail** is not displayed when you traverse using the **detailList-->** bridge. Instead, a list of all categories that have been defined for the object definition appears. This list is labeled **Category**. From each category, you can traverse to a table called **Detail Fields**, which displays all custom fields belonging to the category you selected.

For more information about how custom field values are stored in the database, see Object Model: Custom Fields.

Object: EHistDetail (E_HIST_DETAIL)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
category-->	**CATEGORY_ID (NUMBER)**	**Category**	**x**	**x**	YRecentlyViewedRecord	object	The category of the history with which the details are associated.		
detailDateValueList-->	**n/a**			**x**			Links to the information for history custom fields of type Date.		
detailMemoValueList-->	**n/a**			**x**			Links to the information for history custom fields of type Memo Text.		
detailNumbValueList-->	**n/a**			**x**			Links to the information for history custom fields of type Number and Check Box.		
detailObjeValueList-->	**n/a**			**x**			Links to the information for history custom fields of type List (drop-down or radio button lists).		
detailObjfValueList-->	**n/a**			**x**			Links to the information for history custom fields of type		

							Custom Object.		
detailTextValueList-->	**n/a**			**x**			Links to the information for history custom fields of type Text.		
owner-->	**ENTERPRISE_OBJECT_ID (NUMBER)**			**x**	THistory	object	The history with which the custom fields are associated.		
isManual	**IS_MANUAL (NUMBER)**		**x**			boolean	Specifies whether the category is added manually or automatically to the history. Categories are added automatically when they are the parent of the category that is added manually. 0 - Added Automatically 1 - Added Manually		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	The unique ID of the history category record.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the added category was updated.		

1.1.34.3 EHistUserAccess

EHistUserAccess contains the user access information that is set in the **Security** block of each history record. For example, it includes whether each user that is listed in the Security block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EHistUserAccess (E_HIST_USER_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed		

							or denied. (See allowDenyIID.)		
isPerm	**IS_PERM (NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID (NUMBER)**	**Option**	**x**			char	Indicates whether the user is allowed or denied access to the record. a - Allow d - Deny		
isManual	**IS_MANUAL (NUMBER)**		**x**			boolean	Indicates whether the user rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner -->	**ENTERPRISE_OBJECT_ID (NUMBER)**		**x**	**x**	THistory	object	8 The history record with which this user access setting is associated. --> Links to the specific information about the history record.		
primaryKey	**PRIMARY_KEY**		**x**			int	Unique ID for the user access rights.		

	(NUMBER)								
user-->	**USER_ID** **(NUMBER)**	**List**	**x**	**x**	YUser	object	8 Specifies the user the access rights pertain to. --> Links to the specific information about the user account.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the user access rights have been updated.		

1.1.34.4 EHistGroupAccess

The EHistGroupAccess table contains the group access information that is set in the **Security** block of each history record. For example, it includes whether each group that is listed in the Security block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EHistGroupAccess (E_HIST_GROUP_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ** **(NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE** **(NUMBER)**	**Update** che	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value		

		ck box					is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE** **(NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM** **(NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(NUMBER)**	**Option**	**x**			char	Indicates whether the group is allowed or denied access to the record. a - Allow d - Deny		
group->	**GROUP_ID** **(NUMBER)**	**List**	**x**	**x**	YGroup	object	8 Specifies the group that the access rights pertain to.		

							--> Links to the specific information about the group account.		
isManual	**IS_MANUAL (NUMBER)**		**x**			boolean	Indicates whether the group access rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID (NUMBER)**		**x**	**x**	THistory	object	8 The history record with which this group access setting is associated. --> Links to the specific information about the history record.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	Unique ID for the group access rights.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the group access rights have been updated.		

1.1.35 Object Model: Documents

The tables in this appendix provide information about the object model as it relates to documents.

1.1.35.1 TDocument

TDocument contains general information for a document record, such as the document's author, date, content type, name, file size, version, and so on. This table links to the list of custom fields created for the document categories and related information contained in other tables.

Object: TDocument (T_DOCUMENT)

Attribute	Database	Field in UI	End of	Bridge	Links to object table:	Data	Comments	Commonly used in:

	column name		path			type		Rules	
author-->	**AUTHOR_ID** **(NUMBER)**	**Author** (**General** block)	**x**	**x**	TContact	object	8 The contact who created or uploaded the document file. --> Links to specific information about the user account.		
authorDate	**AUTHOR_DATE** **(DATE)**	**Date Authored** (**General** block)	**x**			date	The date when the file was uploaded. This field can be edited by the user. Time-zone-independent.		
categories--> **Before TeamConnect 3.3 SP2: detailList-->**	**n/a**		**x**	**x**	EDocuDetail In Object Navigator, links to **Category** list which appears in the UI only.		8 The list of categories added to the current object. --> In the object model, the added categories link to the values that have been added for custom fields, according to field type. --> In Object Navigator, this attribute is enhanced. It links to a list of all categories defined for the object definition, rather than those added to the current object. From		

							there, you can select a category and traverse to the list of custom fields that belong to it.		
checkedIn By-->	**CHECKED_IN_BY_ID** **(NUMBER)**		**x**	**x**	YUser	object	8 The user who last checked in the document. --> Links to specific information about the user account.		
checkedIn On	**CHECKED_IN_ON** **(DATE)**		**x**			date	Date when the document was last checked in.		
checkedOutBy-->	**CHECKED_OUT_BY_ID** **(NUMBER)**	**Checked-Out By** (**General** block)	**x**	**x**	YUser	object	8 The user who has checked out the document. --> Links to specific information about the user account.		
checkedOutOn	**CHECKED_OUT_ON** **(DATE)**	**Checked-Out On** (**General** block)	**x**			date	The date when the document was checked out.		
checkOutLocation	**CHECKOUT_LOCATION**		**x**			string (250)	The location where the file was downloaded when it was checked out.		

	(VARCHAR2) (250)								
contentObject--> Before TeamConnect 3.3 SP2: content-->	**CONTENT_ID** **(NUMBER)**			**x**	JDocuContent	object	Links to the content of the uploaded file.		
contentType-->	**CONTENT_TYPE_ID** **(NUMBER)**	**File Type** (**General** block)	**x**	**x**	YDocuContentType	object	8 The file type of the document (For example, Microsoft Excel or WordPerfect). --> Links to the definition information of the document type.		
createdBy-->	**CREATED_BY_ID** **(NUMBER)**	**Created By** (**Security** block)	**x**	**x**	YUser	object	8 The user who created the document record. --> Links to specific information about the user account.		
createdOn	**CREATED_ON** **(DATE)**	**Created On** (**Security** block)	**x**			date	Date and time when the document record was created.		
createdOnBehalfOf	**CREATED_ON_BEHALF_OF_ID**			**x**	TContact	object	When not null, it indicates that the record was created through an e-billing system or other external	**X**	

							application. The contact referenced will often be a vendor in TeamConnect.		
defaultCategory-->	**DEFAULT_CATEGORY_ID (NUMBER)**	• **Default Category** (**General** block) • Indicated by a blue diamond in **Categories** block	**x**	**x**	YRecentlyViewedRecord	object	The default category of the document record.		
groupAccessList-->	**n/a**			**x**	EDocuGroupAccess		Links to the group security information entered through the Security block of the record.		
historyList-->	**n/a**	**History** tab	**x**	**x**	THistory		8 The list of all histories that are related to the document.		

							--> Links to specific history information for all of the document's history records.		
modifiedBy-->	**MODIFIED_BY_ID** **(NUMBER)**	**Modified By** (**Security** block)	**x**	**x**	YUser	object	8 The user who last modified the document record. --> Links to specific information about the user account.		
modifiedOn	**MODIFIED_ON** **(DATE)**	**Modified On** (**Security** block)	**x**			date	The date and time when the document record was last modified.		
name	**NAME** **(VARCHAR2) (250)**	**Name** (**General** block)	**x**			string (250)	The file name of the document. This name can be edited by the user.		
note-->	**NOTE_ID** **(NUMBER)**	**Notes** (**General** block)		**x**	JNote	object	Links to the table that contains the text entered into the **Notes** field of the record.		
parentFolder-->	**PARENT_FOLDER_ID** **(NUMBER)**			**x**	TDocument	object	Links to the information about the parent document folder of this document folder.		
primaryKey	**PRIMARY_KEY**		**x**			int	The unique ID of the document record.		

	(NUMBER)								
securityTypeIID	**SECURITY_TYPE_IID** **(NUMBER)**	**Public/ Private** (**Security** block)	**x**			int	Security Type of Document Record: 0 - Public 1 - Currently Not Used 2 - Private		
shortcutTo Document-->	**SHORTCUT_TO_DOCUMENT_ID** **(NUMBER)**			**x**	TDocument	object	When this document is a Shortcut (specifically, if typeIID = S), links to the document to which the shortcut is pointing.		
size	**DOCUMENT_SIZE** **(NUMBER)**	**Size** (**Documents** screen: Java Applet version only)	**x**			int	File size, in kilobytes (KB).		
treeKey	**TREE_KEY** **(VARCHAR2) (700)**		**x**			string (700)	The chain of primary keys that identifies the parent tree of the file or folder. Used for searching and reporting purposes only.		
type **Before TeamConnect 3.3**	**TYPE_IID** **(NUMBER)**		**x**			int	Specifies whether the document is a: D - Folder		

SP2: typeIID							L - File P - Hyperlink S - Shortcut T - History		
url **Before TeamConnect 3.3 SP2: urlString**	**URL_STRING** **(VARCHAR2) (2000)**	**Create New Hyperlink** Button	**x**			string (2000)	The URL for a hyperlink.		
userAccessList-->	**n/a**			**x**	EDocuUserAccess		Links to the user security information entered through the **Security** block of the record.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the record has been updated.		
versionText	**VERSION_TEXT** **(VARCHAR2) (250)**	**Comments** (**Check In** dialog box)	**x**			string (250)	The text that is entered in the **Comments** field when a user checks in the document. ***Note:*** *This is available to users only when* ***Document Version Control*** *is enabled in*		

							System Settings.		

1.1.35.2 JDocuContent

JDocuContent represents the contents of a document uploaded to TeamConnect, including the actual file and its file extension.

Object: JDocuContent (T_DOCU_CONTENT)

Attribute	Database column name	End of path	Bridge	Data type	Comments	Commonly used in:	
						Rules	Temp/ Wiz
data	**DATA** **(VARCHAR)**	**x**		data	The actual content of the corresponding uploaded file.		
documentType	**DOCUMENT_TYPE** **(VARCHAR2) (50)**	**x**		string (50)	The file extension of the document. ***Note:*** *The preceding period (.) character is added to the file extension, for example, .txt.*		
primaryKey	**PRIMARY_KEY** **(NUMBER)**	**x**		int	The unique ID of the content of a file uploaded to TeamConnect.		
version	**VERSION** **(NUMBER)**	**x**		int	Indicates how many times the file content has been updated.		

1.1.35.2.1 YDocuContentType

YDocuContentType contains information about content types defined for document records. This information includes the name of the document type (such as Text or XML), file extension, and mime type. This table represents the **Document Types** block on the **Document Types** tab of the Document object definition.

Object: YDocuContentType (Y_DOCU_CONTENT_TYPE)

Attribute	Database column name	Field in UI	End of path	Bridge	Data type	Comments	Commonly used in:	
							Rules	Temp/Wiz
fileExtension	**FILE_EXTENSION** **(VARCHAR2) (50)**	**File Extension** (**Document Types** tab of Object Definition)	**x**		string (50)	The file extension of this document type.		
iconFilename	**ICON_FILENAME** **(VARCHAR2) (50)**	**Icon File** (**Document Types** tab of Object Definition)	**x**		string (50)	The name of the image file that should be used in TeamConnect to represent this document type.		
mimeType **Before TeamConnect 3.3 SP2: mimeTypeString**	**MIME_TYPE_STRING** **(VARCHAR2) (50)**	**Mime Type** (**Document Types** tab of Object Definition)	**x**		string (50)	The mime type that should be used for documents of this type.		
name	**NAME** **(VARCHAR2) (50)**	**Document Type Name** (**Document Types** tab of Object	**x**		string (50)	The name of the document type, as it should appear in the Documents screen.		

		Definition)						
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**		int	The unique ID of the document type.		
version	**VERSION** **(NUMBER)**		**x**		int	Indicates how many times the document type has been updated.		

1.1.35.3 EDocuDetail

EDocuDetail contains all categories added to document records. Each time a category is added to a document record, an entry is made in this table in the database.

In Object Navigator, when you list categories, a list of all categories that have been defined for the current object definition appears. This list is labeled **Category**. From each category, you can traverse to a table called **Detail Fields**, which displays all custom fields belonging to the category you selected.

For more information about how custom field values are stored in the database, see Object Model: Custom Fields.

Object: EDocuDetail (E_DOCU_DETAIL)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
category-->	**CATEGORY_ID** **(NUMBER)**	**Category**	**x**	**x**	YRecentlyViewedRecord	object	The category of the document with which the details are associated.		
owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**			**x**	TDocument	object	The document with which the custom fields are associated.		
isManual	**IS_MANUAL**		**x**			boolean	Specifies whether the category is		

	(NUMBER)						added manually or automatically to the document. Categories are added automatically when they are the parent of the category that is added manually. 0 - Added Automatically 1 - Added Manually		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the document category record.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the added category was updated.		

1.1.35.4 EDocuUserAccess

EDocuUserAccess contains the user access information that is set in the Security block of each document record. For example, it includes whether each user that is listed in the **Security** block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EDocuUserAccess (E_DOCU_USER_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ** **(NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected).		

							This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM (NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		

allowDenyIID	**ALLOW_DENY_IID** **(CHAR) (1)**	**Option**	**x**			char	Indicates whether the user is allowed or denied access to the record. a - Allow d - Deny		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the user rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		
owner-->	**ENTERPRISE_OBJECT_ID** **(NUMBER)**		**x**	**x**	TDocument	object	8 The document record with which this user access setting is associated. --> Links to the specific information about the document record.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the user access rights.		
user-->	**USER_ID** **(NUMBER)**	**List**	**x**	**x**	YUser	object	8 Specifies the user the access rights pertain to. --> Links to the specific information about the user account.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the user access rights have been updated.		

1.1.35.5 EDocuGroupAccess

EDocuGroupAccess contains the group access information that is set in the **Security** block of each document record. For example, it includes whether each group that is listed in the **Security** block is allowed or denied the **Read**, **Update**, **Delete**, and **Perm**(ission) rights.

Object: EDocuGroupAccess (E_DOCU_GROUP_ACCESS)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
isRead	**IS_READ (NUMBER)**	**Read** check box	**x**			int	Specifies whether the Read operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isUpdate	**IS_UPDATE (NUMBER)**	**Update** check box	**x**			int	Specifies whether the Update operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
isDelete	**IS_DELETE (NUMBER)**	**Delete** check box	**x**			int	Specifies whether the Delete operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the		

							operations are allowed or denied. (See allowDenyIID.)		
isPerm	**IS_PERM** **(NUMBER)**	**Perm** check box	**x**			int	Specifies whether the Set Permission operation has been selected for granting or denial. Value is 1 (selected) or 0 (not selected). This attribute does not determine whether the operations are allowed or denied. (See allowDenyIID.)		
allowDenyIID	**ALLOW_DENY_IID** **(CHAR) (1)**	**Option**	**x**			char	Indicates whether the group is allowed or denied access to the record. a - Allow d - Deny		
group-->	**GROUP_ID** **(NUMBER)**	**List**	**x**	**x**	YGroup	object	8 Specifies the group that the access rights pertain to. --> Links to the specific information about the group account.		
isManual	**IS_MANUAL** **(NUMBER)**		**x**			boolean	Indicates whether the group access rights were assigned manually through the Security block of the record or automatically by the system. 0 - Assigned manually 1 - Assigned automatically		

owner->	**ENTERPRISE_OBJECT_ID (NUMBER)**		**x**	**x**	TDocument	object	8 The document record with which this group access setting is associated. --> Links to the specific information about the document record.		
primaryKey	**PRIMARY_KEY (NUMBER)**		**x**			int	Unique ID for the group access rights.		
version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the group access rights have been updated.		

1.1.36 Object Model: Common

The tables in this appendix are not related specifically to one of TeamConnect's main T-tables. They have relationships to other objects in the object model and are referred to as common or general objects.

1.1.36.1 JNote

JNote contains the text of notes entered for a record. It is related to all the T-tables except for **TProject** and **THistory**.

Object: JNote (J_NOTE)

Attribute	Database column name	Field in UI	End of path	Bridge	Data type	Comments	Commonly used in:	
							Rules	Temp/Wiz
noteText	**NOTE_TEXT (VARCHAR2)**	**Notes** (**General** block)	**x**		longstring	Contains the text of the notes entered for the record. The TeamConnect user interface enforces a limitation of 20,000 characters for this field.		
primaryKey	**PRIMARY_KEY**		**x**		int	Unique ID for the notes text.		

	(NUMBER)							
version	VERSION (NUMBER)		x		int	Indicates how many times the note record has been updated.		

1.1.36.2 UGrupMember

UGrupMember contains information about a group account's members. This includes the user who is a member of a group and the group to which the user who is a member--essentially, joining the **YGroup** and **YUser** tables. This table represents the **Group Members** block on the **Members** tab of a group account.

Object: UGrupMember (U_GRUP_MEMBER)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
group-->	GROUP_ID (NUMBER)		x	x	YGroup	object	8 The group to which this group member belongs. --> Links to specific information about the user group.	x	
primaryKey	PRIMARY_KEY (NUMBER)		x			int	The unique ID of the group member.		
user-->	USER_ID (NUMBER)		x	x	YUser	object	8 The user who is a group member. --> Links to specific information about the user account.		
version	VERSION (NUMBER)		x			int	Indicates how many times the group member has been updated.		

1.1.36.3 WObjdDetailField

WObjdDetailField contains information about custom fields defined for a category, which includes the field name, default value, field type (such as Text or Number), label, corresponding database name, whether or not it is included in Data Warehouse, whether or not it is required, and category to which the custom field belongs. This table represents the **Custom Fields** block on the **Custom Fields** tab of an object definition.

Object: WObjdDetailField (Y_OBJ_DETAIL_FIELD)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
defaultValue	**DEFAULT_VALUE** **(VARCHAR2)**	**Default Value** (**Custom Fields** tab, Object Definition)	**x**			long String	The default value of the custom field.		
detailFieldTypeIID	**DETAIL_FIELD_TYPE_IID** **(NUMBER)**	**Field Type** (**Custom Fields** tab, Object Definition)	**x**			int	Specifies the custom field type, which is either: 1 - Text 2 - Number 3 - Date Field 4 - Scroll Text 5 - Check Box 6 - List 8 - Involved 9 - Memo Text 10 - Label Only 11 - Custom Object		

extraInfo	**EXTRA_INFO** **(VARCHAR2) (50)**	**Label** (**Custom Fields** tab, Object Definition)	**x**			string (50)	The label of the custom field.		
invlCategory-->	**INVOLVED_CATEGORY_ID** **(NUMBER)**	The list for selecting involved role, if field is of type Involved (**Custom Fields** tab, Object Definition)	**x**	**x**	YRecentlyViewedRecord	object	The involved role selection for the custom field, if it is of type Involved.		
isDataMart	**IS_DATA_MART** **(NUMBER)**	**Include in Data Warehouse** (**Custom Fields** tab, Object Definition)	**x**			boolean	Indicates whether the field is to be included in the Data Warehouse tables.		
isExcludedFromCustomSearch	**IS_EXCLUDED_FROM_CUSTOM_SEARCH** **(NUMBER)**	**Is Excluded from Custom Search?**	**x**			boolean	Indicates whether the custom field is hidden when constructing custom searches, or not.		

		(**Custom Fields** tab, Object Definition)							
isRequired	**IS_REQUIRED** **(NUMBER)**	**Is Required?** (**Custom Fields** tab, Object Definition)	**x**			boolean	Indicates whether the custom field is required or not.		
isTimeZoneIndependent	**IS_TIME_ZONE_INDEPENDENT** **(NUMBER)**	**Is Time Zone Independent** (**Custom Fields** tab, Object Definition)	**x**			boolean	Indicates whether the custom field of date datatype is adjusted in the UI for the end user's timezone (FALSE), or not (TRUE).		
name	**NAME** **(VARCHAR2) (30)**	**Field Name** (**Custom Fields** tab, Object Definition)	**x**			string (30)	The database name for the custom field.		
objectQualifier	**OBJECT_QUALIFIER** **(VARCHAR2) (2000)**	**Qualifier** (**Custom**	**x**			string (2000)	The qualifier path that is used to filter the available records for the		

		Fields tab, Object Definition)					custom field, if the field is of type Custom Object.		
owner-->	**CATEGORY_ID** **(NUMBER)**	**For Category** (**Custom Fields** tab, Object Definition)	**x**	**x**		object	8 The category to which this custom field belongs. --> Links to the definition information of the category.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the custom field.		
searchView	**SEARCH_VIEW_ID** **(NUMBER)**	**Search View** (**Custom Fields** tab, Object Definition)		**x**		object	Available only for custom fields of type Involved or Custom Object.		
table-->	**TABLE_ID** **(NUMBER)**	The list for selecting a lookup table, if the field is of type List (**Custom Fields** tab, Object	**x**	**x**	YDetailLookupTable	object	Specifies the custom lookup table linked to the custom field, if the field is of type List.		

		Definition)							
version	**VERSION (NUMBER)**		**x**			int	Number that indicates how many times the custom field has been updated.		

1.1.36.4 WObjdProjectInfo

WObjdProjectInfo represents a custom project object definition, such as whether the project is contact-centric, embedded, requires a parent project, and various numbering pattern properties. This table links to lists of phases and phase transitions that have been defined for this object.

Object: WObjdProjectInfo (Y_APPLICATION)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
autoNumPattern	**AUTO_NUM_PATTERN (VARCHAR2) (250)**	**Pattern** (**Unique ID** tab)	**x**			string (250)	Indicates the pattern used for auto-numbering.		
autoNumSequence	**AUTO_NUM_SEQUENCE (NUMBER)**	**Next Sequence Number** (**Unique ID** tab)	**x**			int	Indicates the next number in the sequence for auto-numbering.		
childList-->	**n/a**			**x**	WObjdProjectInfo		Links to the child and embedded object definitions for this custom		

							object definition.		
conditionTypeIID	**CONDITION_TYPE_IID** (**NUMBER**)	**Unique ID** tab	**x**			int	Indicates how instances of this object definition will be uniquely identified: 0 - Auto numbering 1 - Pattern of object attributes 2 - Entered manually by user 3 - Default numbering (will be hidden from the user)		
displayFormatIID	**DISPLAY_FORMAT_IID** (**NUMBER**)	**Name** tab	**x**			int	Indicates whether the object definition has been set to use a pattern of attributes for automatic naming. 5 - Allow users to enter name manually 6 - Name constructed from a pattern of attributes		
displayOrder	**DISPLAY_ORDER**	**Order** (**General** tab)	**x**			int	Indicates the order in which the custom object should appear in the		

	(NUMBER)						user interface drop-down lists.		
formatIID	**FORMAT_IID** **(NUMBER)**	Format information in **Unique ID** tab	**x**			int	Indicates how the instances of this object definition should be displayed in the UI: 1 - Name 2 - Number 3 - Name - Number 4 - Number - Name		
isAutoNumEditable	**IS_AUTO_NUM_EDITABLE** **(NUMBER)**		**x**			boolean	Indicates whether users are able to edit the number		
isAutoNumEnabled	**IS_AUTO_NUM_ENABLED** **(NUMBER)**		**x**			boolean	Indicates whether the custom object definition is set to be autonumbered.		
isContactCentric	**IS_CONTACT_CENTRIC** **(NUMBER)**	**Contact-centric** (**General** tab)	**x**			boolean	Indicates whether the custom object definition is contact-centric.		
isEmbedded	**IS_EMBEDDED**		**x**			boolean	Indicates whether the custom		

	(NUMBER)						object definition is embedded.		
isNotDisplayedInMenu	**IS_NOT_DISPLAYED_MENU** **(NUMBER)**	**Do not show this child object definition in the All Services menu** (**General** tab)	**x**			boolean	Indicates whether the child custom object definition appears in the **All Services** menu drop-down lists or is available only through parent records.		
isParentRequired	**IS_PARENT_REQUIRED** **(NUMBER)**	Parent Custom Object Required (**General** tab)	**x**			boolean	Indicates whether instances of the custom object definition require a parent.		
listDisplayTypeID	**LIST_DISPLAY_TYPE_ID** **(NUMBER)**	Options in **List Display** tab of embedded object definition	**x**			int	For an embedded custom object definition, indicates how the embedded object records appear in the UI: 0 - Search view 1 - Batch view		

parent-->	**PARENT_ID** **(NUMBER)**	Parent Custom Object (**General** tab)	**x**	**x**	WObjdProjectInfo	object	8 Indicates the custom object definition that is the parent of this custom object definition. --> Links to information about the parent custom object definition.		
phaseList-->	**n/a**	**Phases** tab		**x**	WObjdPhaseType		Links to the list of phases that have been defined for this object definition.		
phaseTransitionList-->	**n/a**	**Phase Transitions** tab		**x**	WObjdPhaseTransition		Links to the list of phase transitions that have been defined for this object definition.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The primary key of the object definition.		
role	**ROLE** **(VARCHAR2) (250)**	**Role** (**General** tab)	**x**				If the object definition is contact-centric, indicates the label for the contact field.		
rootAssigneeType-->	**ROOT_ASSIGNE**		**x**	**x**	LProjAssigneeType	object	8 Indicates the root node		

	E_TYPE_ID **(NUMBER)**						of the assignee role lookup table. --> Links to information about the assignee role.		
selectionDisplayTypeIID	**SELECTION_DISPLAY_TYPE_IID** **(NUMBER)**	**Default selection mechanism** (**General** tab)	**x**			int	Indicates which selection mechanism is the default for this object definition: 1 - Drop-down list 2 - Search module		
version	**VERSION** **(NUMBER)**		**x**			int	Number that indicates how many times the object definition has been updated.		

1.1.36.5 YDetailLookupItem

YDetailLookupItem contains information about custom lookup table items, such as the item's name, tree position, and display order. The lookup item hierarchy is handled by the attributes **parent-->** and **childList-->**. This table represents the item list displayed in the **Custom Lookup Tables** block on the **Custom** tab of the **Lookup Tables** screen, which are defined to be used for custom fields of type List.

Object: YDetailLookupItem (Y_DETAIL_LOOKUP_ITEM)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz

childList-->	**n/a**			**x**	YDetailLookupItem		Links to the list of all child items that belong to this lookup table item.		
displayOrder	**DISPLAY_ORDER** **(NUMBER)**	**Order** (Lookup Tables, **Custom** tab)	**x**			int	The order in which the item appears in the list.		
name	**NAME** **(VARCHAR2) (50)**	**Item Name** (Lookup Tables, **Custom** tab)	**x**			string (50)	The name of the lookup item.		
parent-->	**PARENT_ID** **(NUMBER)**	**In node:** (drop-down list, Lookup Tables, **Custom** tab)		**x**	YDetailLookupItem	object	Links to the parent item of this lookup table item.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the custom lookup item.		
table-->	**TABLE_ID** **(NUMBER)**	**Show items belonging to** (drop-down list, Lookup Tables, **Custom** tab)		**x**	YDetailLookupTable	object	Links to the lookup table to which this lookup item belongs.		

treePosition	**TREE_POSITION** **(VARCHAR2) (250)**	**Tree Position** (Lookup Tables, **Custom** tab)	**x**			string (250)	The unique four-character alphanumeric code that is created when the lookup item is defined.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the lookup item has been updated.		

1.1.36.6 YDetailLookupTable

YDetailLookupTable contains information about custom lookup tables, such as the name and unique code. This table links to **YDetailLookupItem** for the root lookup item. This table represents the **Custom Lookup Tables** block on the **Custom** tab of the **Lookup Tables** screen.

Object: YDetailLookupTable (Y_DETAIL_LOOKUP_TABLE)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
isActive	**IS_ACTIVE**	**activate** and **inactivate** buttons	**x**			int	Indicates whether the table entry is active (value 1) and can be used when editing records containing that entry. If it is inactive (value 0), records may contain that entry for reporting and historical purposes, but cannot be saved when editing, unless that entry is changed to some other, active table entry.		

name	**NAME** **(VARCHAR2) (50)**	**Show items belonging to** (Lookup Tables, **Custom** tab)	**x**			string (50)	The name of the custom lookup table.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	Unique ID for the custom lookup table.		
rootItem-->	**ROOT_ITEM_ID** **(NUMBER)**			**x**	YDetailLookupItem	object	Links to the information about the root node in the lookup table, such as its name and its child items.		
uniqueCode	**UNIQUE_CODE** **(VARCHAR) (4)**	**Unique Code** (Lookup Tables, **Custom** tab)	**x**			string (4)	Four-character alphanumeric code that uniquely identifies the lookup table.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the lookup table has been updated.		

1.1.36.7 YGroup

YGroup contains information about the group accounts in TeamConnect including the name of the group account and its creation date. This table also links to lists that describe a group's access rights settings, the group's members, tools rights settings, and default object view settings.

Object: YGroup (Y_GROUP)

Attribute	Database column name	Field in UI	End of pat	Bridge	Links to object table:	Data type	Comments	Commonly used in:

								Rules	Temp / Wiz
byPassSearchLimits	**BYPASS_SEARCH_LIMITS**		**x**			boolean	Indicates whether the user account is set to allow users to see the **Return all results** hyperlink in the search results heading. Clicking the hyperlink allows users to bypass the system settings for the maximum number of records retrieved or maximum number of seconds for a search.		
createdBy-->	**CREATED_BY_ID** **(NUMBER)**		**x**	**x**	YUser	object	8 The user who created the group account. --> Links to specific information about the user account.		
createdOn	**CREATED_ON** **(DATE)**		**x**			date	The date when the group account was created.		
functionalAccessList-->	**n/a**			**x**			Links to the group's rights settings.		
users--> Before TeamConnect 3.3	**n/a**			**x**	UGrupMember		Links to the join table that contains information for all		

SP2: groupMemberList-->							members of the group.		
modifiedBy-->	**MODIFIED_BY_ID** **(NUMBER)**		**x**	**x**	YUser	object	8 The user who last modified the group account. --> Links to specific information about the user account.		
modifiedOn	**MODIFIED_ON** **(DATE)**		**x**			date	The date when the group account was last modified.		
displayName **Before TeamConnect 3.3 SP2: name**	**NAME** **(VARCHAR2) (50)**	**Group Account** (**General** tab)	**x**			string (50)	The name of the group account.		
primaryKey	**PRIMARY_KEY** **(NUMBER)**		**x**			int	The unique ID of the group account.		
shortDescription	**SHORT_DESCRIPTION** **(VARCHAR2) (250)**	**Description** (**General** tab)	**x**			string (250)	A description of the group account.		
toolList-->	**n/a**			**x**			Links to the tools rights settings of the group.		
uniqueKey	**UNIQUE_KEY** **(VARCHAR2) (50)**	**Unique Key** (**General** tab)	**x**			string (50)	The alphanumeric code that uniquely identifies the group.	**x**	

version	**VERSION (NUMBER)**		**x**			int	Indicates how many times the group account has been updated.		
viewDefaultList-->	**n/a**			**x**			Links to the default object view settings for the group.		

1.1.36.8 YRecentlyViewedRecord

YRecentlyViewedRecord allows you to define what records appear on the page when the user selects the Recently Viewed Records collection in the left pane.

Object: YRecentlyViewedRecord (Y_ RECENTLY_VIEWED_RECORD)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
primaryKey	**PRIMARY_KEY (NUMBER)**					int	The ID number of the current Recently Viewed record entry.		
recordPrimaryKey	**RECORD_PRIMARY_KEY (NUMBER)**					int	The primary key number of a recently viewed record.		
uniqueCode	**UNIQUE_CODE (VARCHAR2) (4)**					string (4)	The unique code of a recently viewed record.		
version	**VERSION (NUMBER)**					int	The version number of the current Recently Viewed record entry.		
viewedByID-->	**VIEWED_BY_ID (NUMBER)**				YUser	int	The ID number of the user viewing the current Recently Viewed collection.		

viewedOn	**VIEWED_ON (DATE)**					date	The date and time a record was last viewed by the current user whose ID is **viewedByID**.		

1.1.36.9 YUser

YUser contains information about the user accounts in TeamConnect. This information includes the username, user type (such as normal or super), expiration date, password, authentication mechanism, default group, and the associated contact record. It also links to lists that describe a user's access rights settings, group accounts to which a user is a member, tools rights settings, and preference settings.

Object: YUser (Y_USER)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/Wiz
accountExpiresOn	**ACCOUNT_EXPIRES_ON (DATE)**	**Account expires on** (**General** tab)	**x**			date	Date when the user account expires.		
authenticationType	**AUTH_TYPE (VARCHAR2) (10)**	**Authentication** (**General** tab)	**x**			string (10)	The mechanism by which the user is authenticated.		
byPassSearchLimits	**BYPASS_SEARCH_LIMITS**		**x**			boolean	Indicates whether the user account is set to allow users to see the **Return all results** hyperlink in		

							the search results heading. Clicking the hyperlink allows users to bypass the system settings for the maximum number of records retrieved or maximum number of seconds for a search.		
calendarAppUserID	**CALENDAR_APP_USER_ID (VARCHAR2) (250)**	**Calendaring Application User ID**				string (250)	Allows you to map a calendaring server user to a TeamConnect user. User ID of the appropriate calendaring application must be entered. Then, for TeamConnect, you must set: whether the account is active, the expiration date, whether the user must change password at next log-in, and whether to enable assistive technology.		

contact-->	**CONTACT_ID** **(NUMBER)**	**Account belongs to** (**General** tab)	**x**	**x**	TContact	object	8 The contact record associated with this user account. --> Links to specific information about the contact record.		
createdBy-->	**CREATED_BY_ID** **(NUMBER)**		**x**	**x**	YUser	object	8 The user who created the user account. --> Links to specific information about the user account.		
createdOn	**CREATED_ON** **(DATE)**		**x**			date	The date when the user account was created.		
defaultGroup-->	**DEFAULT_GROUP_ID** **(NUMBER)**	Indicated by a blue diamond on **Groups** tab	**x**	**x**	YGroup	object	8 The group that is selected as the default group for this user account. --> Links to specific information about the user group.	**x**	
documentFolder-->	**DOCUMENT_FOLDER_ID** **(NUMBER)**			**x**	TDocument	object	Links to the information about the personal document folder of the user.		

functionalAccessList-->	**n/a**	**Rights** tab **Category Rights** tab		**x**	UUserFunctionalAccess		Links to the user's rights settings.		
groupMemberList-->	**n/a**			**x**	UGrupMember		Links to the join table that contains information for all groups in which this user is a member.	**x**	
isAccountActive	**ACCOUNT_TYPE** **(NUMBER)**	**Account expires on** (check box, **General** tab)	**x**			boolean	Indicates whether the user account is set to expire on a certain date. 0 - false 1 - true		
isActive	**IS_ACTIVE** **(NUMBER)**	**Account is active** (**General** tab)	**x**			boolean	Indicates whether the user account is active. 0 - false 1 - true		
isChangePasswordNextLogin	**IS_CHANGE_PWD_NEXT_LOGIN** **(NUMBER)**	**Must change password on next login** (**General** tab)	**x**			boolean	Indicates whether the user is required to change the password the next time he/she logs in to TeamConnect. 0 - false 1 - true		

isLocked	**IS_LOCKED** **(NUMBER)**	**Account is locked** (**General** tab)	**x**			boolean	Indicates whether the user account is locked after a number of failed login attempts. 0 - false 1 - true		
lastFailedAttemptOn	**LAST_FAILED_ATTEMPT_ON** **(DATE)**		**x**			dateTime	Date and time of the last failed login.		
lastLoginOn	**LAST_LOGIN_ON** **(DATE)**		**x**			date	Indicates the last time the user logged on to TeamConnect.		
ldapStatusIID	**LDAP_STATUS_IID** **(DATE)**		**x**			int	Currently not used.		
longSettingList-->	**n/a**						Currently not used.		
modifiedBy-->	**MODIFIED_BY_ID** **(NUMBER)**		**x**	**x**	YUser	object	8 The user who last modified this user account. --> Links to specific information about the user account.		
modifiedOn	**MODIFIED_ON** **(DATE)**		**x**			date	The date when the user account was last modified.		

numberOfFailedAttempts	**NUMBER_OF_FAILED_ATTEMPTS** (NUMBER)		**x**			int	Indicates the number of failed login attempts.		
parent-->	**PARENT_ID** (NUMBER)						Currently not used.		
password	**PASSWORD** (VARCHAR2) (250)	**Password** (**General** tab)	**x**			string (250)	The password required for the user to log in using this user account.		
passwordSavedDate	**PASSWORD_SAVED_DATE** (DATE)		**x**			date	The date when the password was last saved. This value is used to determine whether the password has expired.		
primaryKey	**PRIMARY_KEY** (NUMBER)		**x**			int	The unique ID of the user account.		
settingList-->	**n/a**			**x**	UUser Setting		Links to the user's preferences settings.		
shortDescription	**SHORT_DESCRIPTION** (VARCHAR2) (250)	**Description** (**General** tab)	**x**			string (250)	Description of the user account.		

toolList-->	**n/a**			**x**			Links to the tools rights settings of the user.		
userTypeIID	**USER_TYPE_IID** **(NUMBER)**	**User Type** (**General** tab)	**x**			int	Indicates the type of this user account, which may be: 1 - Super 2 - Normal 3 - Limited-Privilege		
username	**USERNAME** **(VARCHAR2) (100)**	**Username** (**General** tab)	**x**			string (100)	The username assigned to this user account.		
version	**VERSION** **(NUMBER)**		**x**			int	Indicates how many times the user account has been updated.		

1.1.36.10 YUserSubscribedCollection

YUserSubscribedCollection allows you to define, configure, and set up the display of the types of collections groups of users may subscribe to.

Object: YUserSubscribedCollection (Y_ USER_SUBSCRIBED_COLLECTION)

Attribute	Database column name	Field in UI	End of path	Bridge	Links to object table:	Data type	Comments	Commonly used in:	
								Rules	Temp/ Wiz
contactGroupID	**CONTACT_GROUP_ID (NUMBER)**				YContactGroup	int	The primary key number of the static Contact collection that the user has subscribed to.		

displayOrder	**DISPLAY_ORDER (NUMBER)**					int	The display order of the subscribed collections, expressed as a number, for example, 5th in order = 5.		
primaryKey	**PRIMARY_KEY (NUMBER)**					int	The ID number of the subscribed collection.		
searchID-->	**SEARCH_ID (NUMBER)**					int	The primary key number of the search view that the user has subscribed to.		
userID-->	**USER_ID (NUMBER)**				YUser	int	The primary key number of the subscribing user.		
version	**VERSION (NUMBER)**					int	The version number of the collection subscription.		

1.1.37 Glossary

A | *B* | *C* | *D* | *E* | *F* | *G* | *H* | *I* | *L* | *M* | *N* | *O* | *P* | *Q* | *R* | *S* | *T* | *U* | *V* | *W* |

A

Account

This term can refer to the system object definition called Account, that can be used for tracking various amounts of money allocated for specific purposes.

It can also refer to the TeamConnect records that store the User and Group information, such as name, password, rights and so on. User and Group Accounts can be accessed only by the designated users, such as TeamConnect administrators, on the **Admin** tab.

Action

There are two types of actions in TeamConnect—wizard actions and rule actions.

A wizard page action is an action that is triggered by a page transition in a Wizard. For example, when a user is creating a Record using a wizard, the wizard can do one of the following actions:

- Populate a field in the record with a specified value.
- Execute a Java class file, which can create another record, do a calculation, or do another task automatically.

For more details about wizard actions, see Creating Custom Pages.

A rule action is an action that TeamConnect performs when a Rule is triggered in TeamConnect. Depending on the type of rule, the action that the rule executes could be the following:

- Deny the user the ability to do an operation, such as delete, post, void, update, or create a Record, and display a message to the user.
- Allow the user to do one of the operations above (in other words, do nothing).
- Send the record into a Route of users who can approve or reject the operation attempted by the user.
- Do Custom Actions specified in a JavaScript or Java class file, such as create a record or interact with an outside application (such as Microsoft Exchange).

For more details about rule actions, see the Rule Actions.

Admin Rights

These are rights that control access to all features in TeamConnect that can be accessed by the **Admin** tab in the user interface. These include object definition, lookup tables, routes, home pages, system settings, and so on.

Administrator (TeamConnectAdmin User)

A user whose job responsibilities include creating or maintaining user and group accounts, specifying the appropriate system settings for the organization, creating and/or maintaining system-wide Home Pages and Portal Panes, and making minor changes to the system design if necessary. These responsibilities are reflected by the Admin Rights assigned to this user accordingly.

API

Application Program Interface, a set of routines, protocols, and tools for building software applications.

TeamConnect's API can be used for customizing TeamConnect. You can use the API for purposes such as creating Java rules, integration with other systems, and enhancing the capabilities of various areas of TeamConnect, such as Home Pages.

API Rule

See Custom Action.

Appender

A definition of a system or audit log. Appenders can be of various types, including SMTP, socket and file. File appenders are the most commonly used, as the logging statements are simply captured in a file. A log can be a general TeamConnect log, or it can be associated with a specific logger, so that it captures messages only from a certain TeamConnect area, such as portal panes or custom blocks.

Applet

A small utility written in Java. In TeamConnect, applets can be used for the document screen interface. If applets are enabled on the system, users can enable them for their own use on the **Preferences** tab of the **Preferences** screen.

Application

A term used in the previous versions of TeamConnect to refer to Custom Objects, namely, the objects that are created for your organization.

For more details on custom objects, see Creating Custom Pages.

Appointment

A System Object definition that is intended to allow end Users to track their scheduled events, engagements, or meetings. Appointments are automatically available as a system Block in Object Views for Custom Objects.

Approval

The approval process is the major component of TeamConnect Workflow. Authorization from one or more designated users (known as approvers), which is required for certain actions that are attempted by other users. For example, deleting a matter, posting an invoice, or changing the phase of a matter could require approval and therefore trigger an approval process. When users trigger an approval process, they are considered requestors in the approval process. Users' attempted actions are completed upon their final approval.

Approval Rule

A type of TeamConnect rule, that allows you to define Conditions that when an action requiring an Approval is attempted by a user, it is automatically routed to the designated Approvers. The approval process is defined with the use of a Route.

Approvers

Designated Users who have the authority to approve or reject certain actions attempted by other users who do not have the permission to complete these actions. There can be several approvers at each level of the Approval process. This approval level is also known as an approval Stop, which is a component of a Route. During an approval process, approvers may be able to send the approval request to reviewers for review before they make their decision. Approvers can view their approval requests on the **My Approvals** screen.

Assignees

TeamConnect Users who are assigned to one or more Projects or Tasks. There can be one or several users assigned to a project. In projects each assignee can have a different role, such as attorney, agent, appraiser, paralegal, and so on. Typically, the main assignee is in charge of the whole project, whereas the rest of the assigned users are responsible for certain parts of the project. Tasks can have only one current assignee.

Attendees

TeamConnect Users and members of Contact Groups (Address Books) who are expected to take part in an event, such as a meeting or conference call, for which there is an Appointment scheduled. Designer contact groups are called Address Books in the end-user interface.

Attributes

Building blocks of Objects, which represent data entered by end users. Attributes correspond to columns within tables in the database.

For example, every Custom Object consists of over a dozen attributes, all of which are listed in the main database table for that object: **mainAssignee**, **createdOn**, **CreatedBy**, **ClosedOn**, **defaultCategory**, and so on. The corresponding collection of data constitutes an object record.

You use attributes to identify their corresponding fields when working in various areas of TeamConnect, such as Search Views, Rules, and Wizards. To navigate through selecting the necessary attributes, you can use a tool called Object Navigator. For details, see Using Object Navigator.

Automated Action

An action performed when a rule is triggered that is driven by a Javascript or Java class file. For example, a related Record can be automatically created when the rule is triggered. For more details, see Writing Automated Actions.

Audit Settings

Settings for tracking the actions of users—which screens they access, which selections they make in the screens they visit, when they log in and log out, and other actions.

B

Batch Screen

A type of tab or section in a tab of a record screen where certain multiple data items of the Record can be entered and viewed as a single list, all in one screen. In the object model, items added to a record with batch screens are considered Sub-objects, and are usually represented by J-tables.

An example of a batch screen in the administrative user interface can be the **Custom Fields** tab in object definitions.

Block

A group of system and/or custom fields included into a single file, XML (for custom blocks) or JSP (for system blocks). In the end-user interface, each block is typically displayed as a separate tab or a section on the record screen.

TeamConnect Object Views provided by Mitratech consist of system blocks, such as **General**, **Categories**, **Workflow**, **Assignees**, **Attendees**, and so on. You can also create your own Custom Blocks using Custom Fields.

Business Objects

Representations of the nature and behavior of real world things or concepts in terms that are meaningful to the business. Customers, products, orders, employees, trades, invoices, payments, appointments, accounts, shipping containers and vehicles are all examples of real-world concepts or things that could be represented by Business Objects.

There are System Objects and Custom Objects in TeamConnect.

C

Cache

A special high-speed storage mechanism. The TeamConnect application server has its own user interface caching mechanism to store the screen designs and other user interface data accessed by the application. You can clear the TeamConnect user interface Cache by either re-starting the application server or by clicking **Clear Cache** in the **User Interface Settings** screen of the **System Settings** on the **Admin** tab.

Browsers have their own memory caching mechanisms that can be cleared by closing the browser.

Calendar

A special search results display type available in for search views only in the Appointment object definition. It displays scheduled appointments in a day planner layout.

Cascading Security

The transferring of parent projects' record-level security, automatically in a waterfall fashion, to their related records. This is done on the **Security** tab of only custom object definitions.

Category

Lookup items that allow you to organize custom fields into blocks or sections on the screen for the end-user interface. By adding or deleting categories in a record, the end users display or hide respectively the associated blocks of details in that record. Categories also help users to organize their records by certain types, for example, employee contact records vs. non-employee contact records, domestic vs. international accounts, and so on.

You must specify categories when defining the following:

- Custom Fields
- Blocks
- Access to object records at the category level

Category Rights

See Category Security.

Category Security

The Rights to the object's categories and their custom fields. This is the second level of security, after the object level. All rights assigned at this level take effect only if the access to the object itself is granted.

Category-level rights add more granularity to the object-level rights. A user might have access to the object, but you can still control access to the organization-specific information stored in the custom fields and displayed in custom blocks.

Check In/Out

A way to manage files that are uploaded to the TeamConnect Documents area so that only one person at a time can edit them. If a file is checked out by a user, then no other users can edit the file. When the user is finished editing, he or she then checks in the file, making it available to other users.

Child

Hierarchical relations where one (parent) item can have multiple (child) items, whereas each child item can have only one parent. Typically, parent-child relations can be established between Custom Objects in their object definitions. For example, one Claim can have multiple Features, one Feature can have multiple Sub-features, and so on.

The term "parent" can be also used with reference to the custom object that has Embedded Objects defined for it.

Categories can also have parent-child relations.

Condition

A value that you select for a qualifier in a Rule. For example, a condition could be an assignee role, a phase, a number of days from when the Record was opened or closed, and so on. See also Attribute and Qualifier.

Contact

A System Object definition that is intended to allow end Users to create record for individuals, organizations or separate entities within a organization (such as departments) that deal with their organization. Contacts can be organized into various Contact Groups (Address Books).

Contact Group (Address Book)

A System Object definition that is intended to allow end Users to organize their Contact records together. Users can create groups of contacts with whom they frequently interact, so that their phone numbers are readily accessible when needed. The members of contact groups who are also Users can be added as attendees to appointments.

Contact Groups are called Address Books in the end-user interface.

Custom Action

An action performed by a rule in TeamConnect that is defined in a JavaScript or Java class file. Custom Actions can do a wide range of functions - anything that is possible with Java or Javascript.

Custom Fields

More custom fields that can be created for various objects. These fields allow you to meet the individual needs of your organization. Custom fields differ in each client installation of TeamConnect. When you create a custom field, you specify the label that appears in the user interface and what type of value the field holds (text, number, check box, and so on). You can create custom fields for type of object, system or custom.

Typically, you organize custom fields into Custom Blocks so that you can specify which groups of users can view them.

Custom Object

Business Objects created for your organization. They do not come with the factory default TeamConnect, but are created by a TeamConnect Solution Developer using the object definition area of TeamConnect. Custom Objects can have names used within your industry for files, such as matters, claims, litigation, or policies.

The individual instances of Custom Objects are sometimes referred to as Projects within this documentation.

In TeamConnect 1.6.x, Custom Objects were referred to as "Applications."

For more details on custom objects, see Creating Custom Pages.

Custom View

See Object View.

Custom vs. System

In TeamConnect, various components are referred to as either system or custom. System means that the component is a default component of TeamConnect. Custom means that it is created for your organization. For example, objects, Blocks, fields can be system or custom.

D

DataMart

An older TeamConnect feature providing a repository of data for reporting and analysis. This feature has been superseded by Data Warehouse.

Data Warehouse

An optional TeamConnect product. A repository of data gathered from operational data and other sources that is designed to serve a particular community of knowledge workers. In scope, the data derives from an enterprise-wide database, such as TeamConnect database, and meets your specific demands in terms of analysis, content, presentation, and ease-of-use. Typically, Data Warehouse is used for general reporting purposes.

If you are using Business Objects™ or other software to run reports and you want to include the information entered in a Custom Field for your reports, then you must make this selection when creating the custom field in an object definition.

Debug Settings

A tool that allows you to select whether you want TeamConnect to log various program operations.

Default

A value or setting that TeamConnect is automatically selects if the user does not specify a substitute. For example, TeamConnect might select certain Categories as default categories for Records, or set one of a contact's addresses as default.

Likewise, the default directory (for example in documents) is the directory the operating system searches unless you specify a different directory.

The default can also be an action that the program takes. For example, a user who creates an appointment is automatically (or by default) added as an attendee for this appointment.

Document

A System Object definition that is intended to allow end Users to manage files uploaded to the Documents area of TeamConnect. These can be plain text, Microsoft Word, Excel, PDF, image (JPEG, GIF, BMP), HTML, XML, and so on. They are usually attached to TeamConnect Records.

Documents are automatically available as a system Block in all Object Views.

Many files necessary for customization of TeamConnect are also uploaded as Documents. These include XML files for Custom Blocks, Custom Actions for Java Rules, files included in Templates, and other files related to your customized system design.

Document Type

File type. For example, Microsoft Word document (DOC), text file (TXT), or JPEG image (JPG). When uploading files into the Documents area of TeamConnect, you specify the document type. TeamConnect then recognizes that file type and displays the appropriate icon.

As a solution developer, you can specify which file types TeamConnect is able to identify in the Documents area by adding them to the list of Document Types in the Document object definition.

Dynamic Value

A value that is automatically generated based on a value that is entered or selected in another field, as opposed to a Static Value. Dynamic values are often used in TeamConnect Templates, Wizards, Unique IDs and names for custom object Records.

E

Entity

This term is sometimes used to refer to Objects in TeamConnect.

End User

The final or ultimate User of the system. The end user is the individual who uses TeamConnect after it has been fully developed and designed, by way of the user interface. This term is typically used when the distinction from the users with administrative rights needs to be emphasized. It usually implies an individual with a relatively low level of computer expertise. Unless you are a programmer, engineer, or solution developer, you are likely an end user.

Embedded Objects

Simplified Custom Objects created within other (parent/main) Custom Objects that can only exist within the context of the parent Custom Object. Within Wizards and Templates, embedded objects are considered to be Related Objects.

Expense

A System Object definition that is intended to allow end Users to track internal costs of doing business in their organization. Expenses are automatically available as a system Block in Object Views for Custom Objects.

F

Friendly Name

A label that you give to a TeamConnect component, such as a Custom Block, in the form of a text name instead of a code. Friendly names allow you to identify the component among the other components with which it is listed in the screen, so that you do not have to remember a unique code or a key that is automatically assigned by TeamConnect.

Full-Text Search

Allows you to search for items in the database according to their content. TeamConnect provides the feature of content searching in uploaded documents, and certain fields in Records, if Microsoft® SQL Server Full-Text Search or Oracle® Text (interMedia) is available on your database server. For example, using content searching, you could search the content of uploaded documents to find a file in which a certain person's name is mentioned. This feature is available in search screens if the appropriate fields are included in the corresponding Search View.

Functional Level Security

Discontinued term. Replaced by the concept of object-level Rights.

G

Group

Refers to either Contact Groups (Address Books) or User Groups.

User groups are certain groups of users who have the same or similar responsibilities within the organization. Each User Group has its own set of access rights assigned to it. Groups can also have different Object Views.

Group Rights

The Rights to all features that can be accessed in the end user interface. These primarily include the rights to all System Object and Custom Object records, and certain Tools, such as Scheduler, Batch Task and Expense Entry. Rights are assigned by user group.

H

History

A System Object definition that is intended to allow end Users to make chronological entries always associated with a specific Record. History is a Related Object only; that is, History records can only exist when related to other records. History is automatically available as a system Block in Object Views.

Home Page

The first screen that users see after logging in to TeamConnect. Home Pages function as a starting point for all TeamConnect users and can be personalized to fit their individual needs. Users can access various types of information or do different actions from Portal Panes that are available on the Home Page.

I

Invoice

A System Object definition that is intended to allow end Users to track invoices or bills sent by Vendors who provide their organization with goods or services. Invoices are automatically available as a system Block in Object Views for Custom Objects.

Involved

A System Object definition that is always related to a Custom Object. It is intended to allow end Users to keep track of various involved parties associated with a Project. Involved is automatically available as a system Block in Object Views for Custom Objects.

L

Line Item

A System Object definition that is intended to allow end Users to list the goods or services specified in a vendor Invoice. The details of each line item, such as the type (fee or expense), price, quantity, and dates must be entered in the Line Items screen of the corresponding invoice. Line Items are automatically available as a system Block in Object Views for Custom Objects.

Logger

A component for a specific area of TeamConnect for which a system or audit log can be defined. For example, TeamConnect has loggers for portal panes, custom blocks, the XML layer, and user login/sign off. When you define a log (or appender), you select which logger's messages it captures.

Lookup Table

Tables that organize, store and quickly access multiple items, such as contacts' roles, types of addresses, phone, fax and other contact information, and so on. In addition to adding, renaming, inactivating, and deleting items in system lookup tables, you can define new custom lookup tables and their contents. In the end-user interface, lookup tables are represented by either drop-down lists (most commonly) or radio buttons, where the user can look up the necessary information and make the appropriate selection.

M

Member

This term can refer to the following:

- Group members are Users who are included in a group, from which they can obtain rights to various objects and tools in TeamConnect.
- Route members are users who are included in the Approval process in an Approval Rule.
- Contact Group (Address Book) members are contacts that are included in a user's personal Contact Groups.

Memo Text

A type of text field in TeamConnect in which you can type free-form text of a length up to what your database permits.

N

Node

Different levels in the Lookup Table tree structures that have their own sub-levels.

Null Value

An empty or blank value in a field. For example, a field has a null value if the user has not entered anything into the field. It is important to understand this term when writing rules because the Rules screen sometimes indicates that there is a null value selected in a qualifier by displaying **(null)**. The `allowNullValue` tag can also be used for custom fields of type Drop-down List.

O

Object

Within this documentation, it is the same as Business Object.

This term also overlaps the object-oriented programming terminology, where an object is defined as a self-contained entity that consists of both data and procedures to manipulate the data.

Object Attribute

See Attribute.

Object Definition

A specification of all properties of a System Object or Custom Object. After fully defined, it can contain all details regarding the object, from its name, icon, categories, and custom fields, to the way its record and search screens look, wizards that can be used for creating its Records, naming and numbering patterns for (custom object) records, and rules that are triggered when users are working with the object records.

Object-Level Security

See System Record Rights and Custom Record Rights.

Object Model

The architecture in which data exists in TeamConnect. In the user interface, it is represented in the form of tables that hold information about objects and Records in the form of Attributes. Object Navigator allows you to navigate through these tables to select attributes when you are creating Rules, Wizards or Templates, or possibly when you are defining the naming convention of an object. The object model is fully documented in Object Model: Read This First and the additional reference tables it points the user to.

Object Navigator

A tool that allows you to traverse through the tables of the Object Model to select Attributes. This is necessary when creating rules, templates, wizards, and possibly when defining the naming convention of an object. Using Object Navigator, you create a Path that identifies an Attribute. For details, see Using Object Navigator.

Object Table

A table in the Object Model that contains a list of Attributes belonging to a specific object.

Object View

A specific layout of tabs, sections, blocks, and other user interface elements on the screen that can be assigned to users or groups, which you can customize by adding/removing system and custom Blocks in the tabs as desired. In each object definition, several Object Views can be created so that different groups of users see the information that is relevant to them.

Operator

A specific action that allows you to manipulate and assign values to fields in templates and wizards. For example, **set to value of path** sets the value of a field to the value specified by the Path.

P

Parameter

A wizard page component that allows you to add certain data items that you might need to use as qualifiers in rules when creating conditions for page transitions. You can also use parameter values to do certain Actions (or operations) on the object attributes. These data items are not stored in the TeamConnect database and can be added and used only within Wizards.

Parent

See Child.

Path

A series of Attributes that is created using Object Navigator. A path is used to identify how to get to a specific attribute in a table of the Object Model. Paths can be used in Rules, Templates, Wizards, and in the naming patterns for object records.

Phases

Certain states of a Project. A specific sequence of phases and their transitions define the life cycle of the project. For example, a Litigation record might go through the following phases Suit Filed > Discovery > Depositions > Settlement > Arbitration >Trial > Closed. All Phases and their transitions are defined in the Custom Object definitions.

Portal Pane

A component of a Home Page, which represents a specialized menu that allows the end users to access different information or do specific tasks directly from the Home Page. includes one or more items of content.

Preferences

System options that you can use to customize your TeamConnect to fit your individual needs and personal preferences. They are listed when you click the **Preferences** link in the user interface.

Primary Key

An automatically generated number that uniquely identifies the newly created item in the database.

Primary keys are rarely displayed in the record screens. However, they can be used when building paths with Object Navigator, for example in rule qualifiers, in custom object record names and unique identifiers, or when working with the XML layer.

Process Manager

A TeamConnect user who is responsible for monitoring approval processes that are sent to approvers by requestors. Process Managers receive notifications of approval requests when there is an error. They may also be notified when a request is rejected or expired. They can restart an approval process to resolve errors, rejections, or expirations, or reassign the task of approving a request to other users when necessary. They can also reject approval requests.

Project

Is a generic term used to refer to an individual Record or instance of a Custom Object.

Q

Qualifier

Rule qualifiers are criteria used to determine whether a user's attempted operation, such as deleting a Record, should be permitted. Qualifiers can include who the user is, the phase of the record, the number of days from when the record was closed, and countless other possibilities.

Search View qualifiers are Attributes used as criteria that a user can use to search for records, such as the name of the record, the record number, the phase of the record, the record's assignees, and countless other possibilities.

Quick Search

Search performed directly from a Search Module field by typing search criteria in the field and then clicking the **Find** icon. The formatting requirements for typing your criteria vary depending on whether the Search Module field is for a Project, Contact, User, or Account.

R

Read-only Access

Record-level security state in which the record cannot be modified as a result of the rights assigned to it or for workflow reasons, when the record is pending approval, and the approval rule prevents anyone from modifying the record. This relates to Record Security and not to the read-only display state.

Record

An individual instance of a System Object or Custom Object in TeamConnect, created and accessed in the user interface. Note that Custom Object records are sometimes generically referred to as Projects.

Record Security

Access protection set at the level of individual object records. For example, you can make a Record Private or Public or you can grant or deny the rights to read, update, and delete a record to other Users or Groups. This level of security can be set only on the **Security** tabs of individual records in the end-user interface. This is the lowest level of access rights. All rights assigned at this level take effect only if the appropriate rights are assigned at the object level.

Rejection

Denial of an Approval request to do a user-attempted operation by members of a Route. If the operation is rejected, this is noted on the **Workflow** tab of the Record.

Related Objects

Objects that are in one way or another are associated with other objects in TeamConnect, either using hierarchical relations (child-parent, Involved, Embedded Objects), or non-hierarchical relations established in Custom Fields of type Custom Object.

Request

An attempted operation by a user that is not completed until it has been approved by the approvers in a specified approval route. A request can be tracked by the requestor, the approvers who receive the request, and possibly a Process Manager, if one is specified for that approval process.

Requestor

A user who performs an action that triggers an approval process, such as posting an invoice, voiding an invoice, or changing the phase of a project (such as a matter). The request is sent to approvers according to the approval route that is defined for that workflow process. The requestor's request is approved or rejected by the approvers. Requestors can view the progress of their requests on the **My Requests** screen. They can also cancel a request if necessary.

Required Fields

Fields that must have a value entered in them because a Record cannot be saved. Depending on the end-user TeamConnect configuration, required field labels might be identified in the user interface in red (or other color different from the default text color), or other special formatting applied to the label text, such as **bold** or *italics*.

You can make Custom Fields required by specifying them as such when you create them. You can also make fields required only under certain conditions, with the use of Rules.

Resources

Resources are means and/or facilities necessary for conducting an event, for which an Appointment has been scheduled. For example, you might need to book a particular conference room or a projector.

Reviewer

A user who is asked by an approver to review an approval request. Reviewers are not approvers, so they do not approve or reject the request. They can provide feedback to the approver who makes a decision in the process. Reviewers can view their requests for review on the **My Approvals** screen.

Rights

Privileges or permissions to use information that taken together constitute security at different levels, object, category, and individual Record. In TeamConnect, access rights ensure that no record information can be read and/or compromised by unauthorized individuals. Typically, rights are assigned by the TeamConnect system Administrator to individual Users and/or Groups.

Root

The top Category in the categories hierarchy tree. It is always automatically added to every record in TeamConnect. In Projects, this category has the same name as the object itself.

Route

A defined hierarchy of users whose Approval is required to authorize a requested operation. It is an approval path that can consist of one or more nodes, called Stops, that can contain one or more Members who can approve or reject the operation. Routes are used for the Workflow purposes, to control the actions that take place in your business processes.

Rules

Business Workflow requirements that allow you to depend on TeamConnect to ensure that the proper processes, which are specific to your industry and organization, are maintained by users.

TeamConnect allows you to create rules that control whether the operations attempted by users are in accordance with your business processes. A rule is triggered when the operation is attempted, and the Qualifiers are checked to determine what should occur. For example, rules can ensure that certain Custom Fields are filled out if a particular Category is selected.

Approval rules postpone the operation until all members in the approval process, or Route, have approved the operation.

Rules written using Java class files or JavaScript can also perform Automated Actions when they are triggered.

S

Search Module

A special utility that allows the user to quickly search for a specific record and link it to the record with which you are currently working without leaving its screen. The Search Module is represented by a field with two icons, **Find** and **Open**, next to it.

Search Screen

A type of screen that appears whenever you are looking for a specific record. Search screens consist of two tabs: **Filter** and **Results**. In the user interface search screens can be displayed as separate screens, tabs, or sections of a tab. You can access these screens if you select a provided search option or if you click the **Find** icon in record screens. You can define what types of search screens are available to users by creating Search Views.

Search Views

Specific sets of search Qualifiers and search results display settings that you can define in each object definition to allow end users to quickly find the necessary object Records and/or create quick reports. Typically, there are two default Search Views each object definition: **Default** and **Advanced**. However, you can create customized search views (such as **My Tasks**, **Recent Invoices**, and so on).

Security

See Rights.

Security Rule

A type of TeamConnect rule that you can create to check the specified Qualifiers before allowing or denying the user's attempted operation. A security rule is based on the conditions in the Record when it was first opened by the user, so that the user cannot change the values of fields if he or she should not have permission. For more details on security rules, see Security Rules. See also Validation Rule.

Smart Search

A search utility located in the user interface. Smart Search allows users to search for specific record types or across **All Records** or **All Projects**. Users can enter a few consecutive characters from any record's name or number, select the type of record, and the system will figure out what to search by, name or number, and whether the characters come from the beginning, middle, or end.

SQL

Structured Query Language. Used for performing operations in a database, such as searching, adding values, and altering values.

Static Value

A value that the user manually enters or selects. Unlike Dynamic Values, static values are not based on the values in other fields. It is important to understand the difference when working with TeamConnect Templates, Wizards, Unique IDs and names for custom object Records.

Stop

An Approval node, or check-point in a Route within the TeamConnect Workflow. Each stop consists of a certain number of Approvers, whose authorization is required for a pending action. Each approval route might have several stops.

Sub-object

In end-user interface, it is an item that is typically added to an object Record using a Batch Screen. Different objects have different sub-objects. For example, the following items are all sub-objects:

categories, task assignees, project assignees, project relations, contact relations, and the addresses, emails, skills, phone numbers and other items added to contact records.

In the TeamConnect object model, sub-objects are always represented by J-tables, such as JProjAssignee, JContRate, JApptAttendee, and so on. The only exception is categories that are always represented by the WObjdCategory object table.

Super User

A user who can access private and public Records regardless of the settings on the **Security** tab of these records, provided this user has the object-level Rights to these records.

System vs. Custom

In TeamConnect, various components are referred to as either system or custom. System means that the component is a predefined component that is installed with TeamConnect. Custom means that it is created for your organization. For example, objects, Blocks, and fields can be system or custom.

Each object definition in TeamConnect has its own system fields, organized into system blocks that are added to its system views. All system views are automatically set as default system-wide object views.

System Object

Default Business Objects provided by Mitratech to your organization. They represent business objects that are common to most business models regardless of the specific industry.

There are 13 system objects within TeamConnect: Account, Contact, Contact Groups (Address Books), Appointment, Expense, Invoice, Task, History, Document, Involved, User account, and Group account.

System Settings

Settings, such as password policy, user interface settings, and so on that affect the whole system. Typically available to TeamConnect Administrators.

T

Tab

A user interface element that appears at the top of all user and administrative record screens. When clicked each tab opens a new set of information from the record.

In the end-user interface, all tabs in object Records are defined using the Search Views in the object definitions. When you define tabs, you must add at least one Block in order for the tab to be displayed on the screen.

Table

See Lookup Table, Object Table.

Tag

An instruction that is included in an HTML or XML file. The instruction is enclosed between the greater than (>) and less than symbols (<), also known as angled brackets. You use tags when creating Blocks, or Document Templates.

Tag Attribute

A value that modifies an HTML or XML tag. For example, if you want to specify that a number field is formatted to display as a dollar amount in a form, you would use the attribute `format="DOLLAR"`.

For a list of available tag attributes for TeamConnect Blocks, see Using Tag Attributes section.

Task

A System Object definition that is intended to allow end Users to track internal assignments that they have to do, usually while working on Projects, for example preparing reports, following up on cases, reviewing documentation, and so on. Tasks are automatically available as a system Block in Object Views for Custom Objects.

Template

This term can refer to the following:

- Document templates
- Templates used with Wizards
- Templates used with rules
- Templates used with wizards and rules

A document template is a template designed for use with the Document Generator. You can use this capability to generate documents from Records using certain fields, such as the name of the contact associated with the record.

Templates used with wizards or rules define default values that should be automatically filled in for certain fields, Sub-objects, and Related Objects. For more details on templates, see Creating Custom Pages.

Tool

A utility that helps to automate an operation to decrease the time and effort that would be expended on a task, such as reassigning work in bulk to various Users. There are system tools that come by default with TeamConnect; however, it is also possible to create custom tools. For details about creating custom tools, see Creating Custom Tools.

Transaction

Events or happenings in your business that change its financial position, such as buying supplies, paying bills, withdrawing money, or buying equipment. For example Expenses, Tasks, and Invoices and regular money withdrawals from and transfers between Accounts incur transactions against accounts.

Traversing

The act of jumping from one table to another in Object Navigator. This is accomplished by selecting one of the Attributes that appears with an arrow next to it. Each time you traverse to another table, you add the selected attribute to the Path which you are creating to identify a field in TeamConnect.

Tree Position

The last branch in a lookup table hierarchy tree, represented by 4-character alphanumeric combination assigned to an item listed in a Lookup Table, a Category or an Assignee Role. Tree positions uniquely identify these items in TeamConnect database. They can include letters and numbers, but letters used in a tree position must be capitalized.

The full tree position path of each child lookup item includes the tree positions of all of its parents.

U

UBB code

Ultimate bulletin board code--a variation of HTML formatting tags that allows you to add functionality and styles to your messages.

Unique Code

A unique code that identifies a system or custom Object in the TeamConnect database. It consists of a 4-character, alphanumeric combination, and its letters must be capitalized. System objects have predefined unique codes, whereas custom objects must have unique codes defined when first created.

Phases have 4-character, alphanumeric combination assigned to them too, which are also knows as unique codes.

Unique ID

Identifies an individual custom object record. You must set unique IDs in the object definition of each custom object. For example, you can set the records to be automatically numbered by the system, or to be given a unique id based on a pattern of Attributes that are selected in the record.

Do not confuse the Unique ID for Custom Object records with the Unique Code of the Custom Object. Also, do not confuse it with naming patterns for Custom Object records.

User

An account created for a specific person who uses TeamConnect. Users can be members of Groups, and can have Rights assigned to them or from a group in which they are a member. Each user has their own password to log in to TeamConnect, which is first specified in their user account (accessible by the Administrator only) together with the username and rights.

User Interface

The graphical representation of TeamConnect on the computer screen, where the user can click buttons and hyperlinks, make selections from drop-down lists, enter text into fields, and so on. It is also referred to as the graphic user interface or GUI. The end-user interface in TeamConnect includes all available TeamConnect features, because end users do not typically have access to items in the **Admin** tab.

User-Invoked Actions

See Custom Actions.

V

Value Set

A group of attributes in a Sub-object which end users must always populate to make a valid sub-object entry. When creating Templates, you must make sure that you that all attributes from the sub-object's value set are defined. For more details on defining sub-objects in templates, see the Creating Custom Pages.

Validation Rule

A type of Rule in TeamConnect that you can create to check the specified Qualifiers before allowing or denying a user's attempted operation, such as creating or updating a Record. For more details on validation rules, see the Validation Rules. See also Security Rule.

Vendor

Outside Contacts with whom your organization conducts business. These can be suppliers of goods or services, outside counsel, and so on.

Version Control

A method of tracking the edits made to a document that is uploaded to TeamConnect. Version control allows users to access previous versions of a document, and even revert the current version back to a previous version if needed. Version control must be activated by the TeamConnect Administrator.

W

Wizard

A utility that you can create allows users to create object Records. A wizard guides the user through pages of questions and instructions, and, if you have a Template associated with the wizard, automatically enters values into the correct fields. A wizard can even do actions such as automatically creating related records, performing calculations, and other automated actions.

Workflow

Workflow is a sequence of activities within an organization to produce a final outcome.

For example, in a financial setting, a request to delete an account might be automatically routed from a financial officer to accounting manager to accounting director for approval and back to the officer for processing.

At each stage in the workflow, one User or Group is responsible for a specific operation. After the operation is complete, TeamConnect ensures that the users responsible for the next operation are notified and receive the data they need to execute their stage of the process.

1.2 Legal Customization Help

Welcome to the *TeamConnect® Legal Matter Management Customization Help*.

File Naming Conventions	Time Entry Tool
CSM Configuration for Legal	SOP Configuration for Legal

1.2.1 Legal Customization Guidelines

Important: *Mitratech recommends that the shipped design files remain untouched.*

When customizing TeamConnect® Legal Matter Management, you should follow these general guidelines:

- If you modify any files that are delivered with TeamConnect® Legal Matter Management, you must not use the same file names. Instead, copy and rename the files. This will help distinguish between problems with the modifications and problems in the delivered product.
- A Java developer should make any necessary modifications to copies of TeamConnect® Legal Matter Management's Java class files.
- Renaming TeamConnect® Legal Matter Management components usually will not cause problems, but you should not change their tree positions because many rules and wizards depend on them.
- Hiding or deactivating a component is preferable to deleting it. For example, rather than delete unneeded blocks, remove them from the object view.
- Before deleting a component, determine whether any class files, XML files, rules, or wizards depend on it so that you can make corresponding modifications to the dependent components. You can search the source code for the unique key of the item to determine whether any class file rules or wizards depend on the item.

  You must have access to the Java source code to be able to search for such text strings.

1.2.2 File Naming Conventions

This section explains the file naming conventions for the Java class, XML, and JavaScript files that are shipped with TeamConnect® Legal Matter Management. These conventions provide clues as to the function of the files. This section explains the file naming conventions so that you can identify files that you may want to copy, rename, and then modify to suit your business needs.

The files delivered with TeamConnect® Legal Matter Management all use the **SYS** suffix. In order to distinguish the files that were shipped with the product from files that have been customized, solution developers must not simply modify the delivered files, but should at least change the **SYS** suffix to another string--for example, **IMP**.

Caution: *If you modify any files that are delivered with TeamConnect® Legal Matter Management, you must not use same file names. Instead, copy and rename the files.*

Solution developers can create their own naming patterns. However, Mitratech recommends that they follow the general file naming conventions so that the function of the corresponding Java class, XML, and JavaScript files will be implied by their file names.

1.2.2.1 Filename Parts

TeamConnect® Legal Matter Management file names can include some, but not necessarily all, of the following components in the following order:

PrefixComponentMatter_ObjectDescriptionTrigger/QualifierSuffix.FileExtension

The parts of the filename are delimited with capital letters. For example, the file **CommonWzMatterPopulateCategoriesQSYS.class** has the following parts:

- **Common—**The Common prefix indicates that this file contains a super class that is inherited by more than one object.
- **Wz—**The **Wz** component indicates that this file is used by a wizard.
- **MatterPopulateCategories—**This descriptive string indicates that the file is used to populate categories for matter records.
- **Q—**The **Q** indicates that this file acts as a qualifier.
- **SYS—**The **SYS** indicates that the file was delivered with TeamConnect® Legal Matter Management and was not developed during or after implementation.
- **.class**—Indicates a Java file.

Prefixes

The following filename prefixes are used with the **SYS** files:

- **Common**—Indicates that the file contains a super class that is inherited by more than one object.
- **TCL**—Indicates that the file contains utilities used by various class files.

For example, **TCLActionHandlerSYS.class** contains an action handler utility used by other class files.

Components

The following filename components are used with the **SYS** files:

- **Cjb**—Indicates that the file contains a custom Java block (CJB).
- **Rule**—Indicates that the file is used by a rule. This also indicates that the file is not associated with a wizard.
- **Wz**—Indicates that the file is used by a wizard.
- **WzCjb**—Indicates that the file is a CJB for a wizard page.

For example, **RuleTCreateBudgetCASYS.class** is used by a rule for creating budgets upon the creation of a Transaction record, but the file is not directly associated with a wizard.

Matter Object

The following letters indicate the matter object to which the **SYS** files apply:

- **AC—**Advice & Counsel
- **D—**Dispute
- **T**—Transaction

For example, **RuleTUpdateBudgetUASYS.class** contains a rule for the Transaction object that updates a budget upon the Update action.

Description

The description is usually an intuitive string that indicates the file's functionality. For example, the string may be as simple as **Budget** or include a description of a qualifier, such as **IsNoFirmsSelectedAndFirmsListEmpty**. If the string is prefixed by **Matter**, the qualifier or action affects more than one matter object.

The following acronyms appear in the descriptive string of some filenames:

- **AC—**Advice & Counsel
- **IP—**Intellectual Property
- **OC—**Outside Counsel
- **OP**—Opposing Party

For example, **WzTValidateFieldsQSYS.class** contains qualifiers used by the Transaction object's New Transaction wizard to validate fields.

Trigger/Qualifier

The following acronyms indicate triggers or qualifiers in the **SYS** files:

- **A—**Action (most likely for a wizard action)
- **CA—**Performs an action on Create
- **CUA—**Performs an action on Create or Update
- **DA—**Performs an action on Delete
- **Q—**Qualifier
- **AQ—**A qualifier for an Approval rule
- **UA**—Performs an action on Update

For example, **CommonWzOutsideCounselBudgetASYS.class** contains a common wizard action for the creation of outside counsel budgets.

Note: *A file that does not have a prefix in its filename or a trigger/qualifier before the **SYS** suffix is probably a user interface design XML file that does not use Java. For example, the **LawsuitDetailsSYS.xml** file is not prefixed by **Common** or **TCL** and does not have an **A** or **Q** before **SYS**.*

1.2.2.2 Utility Class Files

The following utility class files perform the corresponding function:

- **ConstantsSYS.class—**Holds the system constants in TeamConnect® Legal Matter Management. Stores tree positions, custom field names, wizard parameter names, etc.
- **TCLAccountUtilitySYS.class—**Provides alternatives to the standard TeamConnect Enterprise account-processing methods. Holds transaction, deposit, and withdrawal methods that provide alternatives to the standard TeamConnect Enterprise account-processing API.

Caution: *To avoid account-processing errors, you should carefully plan any modifications to the functionality contained in the* ***TCLAccountUtilitySYS.class*** *file.*

- **TCLBudgetUtilitySYS.class—**Holds the budget constants and utilities in TeamConnect® Legal Matter Management.
- **TCLFacadeSYS.class—**Stores many utility methods used by TeamConnect® Legal Matter Management.
- **TCLParameterTokenizerSYS.class—**Holds lists of parameters that are created during execution. This class provides methods to generate and parse one long string that holds all the parameter data. This class file uses a corresponding JavaScript file called **TCLParameterTokenizerSYS.js**.
- **TCLSecurityUtilitySYS.class—**Stores security- and access-management-related utility methods.

1.2.2.3 Base Classes

The following base class files provide enhanced functionality in CJBs:

- **TCLActionHandlerSYS.class—**An abstract super class that stores essential utilities shared by all action classes.
- **TCLConditionHandlerSYS.class—**An abstract super class that is used in qualifiers and stores essential utilities shared by all condition classes.
- **TCLCustomBlockSYS.class—**An abstract super class used by all custom block classes and stores essential utilities shared by all CJBs.

1.2.2.4 Batch Display Files for Custom Java Blocks

Most of the CJBs in TeamConnect® Legal Matter Management 3.4 use one XML file and one class file. However, CJBs with batch displays require two additional files to define the new and regular rows of the batch display. For example, the **Cost Center** block for the Transaction object uses the following four files:

- **CjbTCostCenterSYS.class—**Java class file
- **CjbTCostCenterSYS.xml—**Primary XML CJB design file
- **CjbTCostCenter_NewRowSYS.xml—**XML design file that represents the New Row
- **CjbTCostCenter_RegularRowSYS.xml—**XML design file that represents the Regular Row

The same convention is used for the **Cost Center** blocks for the Dispute object:

- **CjbDCostCenterSYS.class—**Java class file
- **CjbDCostCenterSYS.xml—**Primary XML CJB design file
- **CjbDCostCenter_NewRowSYS.xml—**XML design file that represents the New Row
- **CjbDCostCenter_RegularRowSYS.xml—**XML design file that represents the Regular Row

The batch display convention is also used for the **Environmental Corrective Action** block, which is displayed on the **General** tab of Dispute records that have the Environmental category:

- **CjbDCorrectiveActionSYS.class—**Java class file
- **CjbDCorrectiveActionSYS.xml—**Primary XML CJB design file
- **CjbDCorrectiveAction_NewRowSYS.xml—**XML design file that represents the New Row
- **CjbDCorrectiveAction_RegularRowSYS.xml—**XML design file that represents the Regular Row

The batch display convention is also used for the **Real Estate Lessee** block of the Transaction object, which is displayed on the **General** tab if the Real Estate category is selected. The **Real Estate Lessee** block uses the following files:

- **CjbTLesseeSYS.class—**Java class file
- **CjbTLesseeSYS.xml—**Primary XML CJB design file
- **CjbTLessee_NewRowSYS.xml—**XML design file that represents the New Row
- **CjbTLessee_RegularRowSYS.xml—**XML design file that represents the Regular Row

1.2.3 Object Definitions

This section provides TeamConnect® Legal Matter Management-specific guidelines regarding the following activities:

- Changing the name of an object definition
- Modifying object definition attributes
- Configuring object definition naming patterns

Performing these procedures is covered by the TeamConnect Enterprise Solution Developer documentation.

Changing the Name of an Object Definition

You can change the name of a custom object definition without causing any problems because no rules or wizards depend on object definition names. However, you must not change the tree position of a custom object because many rules and wizards depend on the tree position.

Modifying Object Definition Attributes

You can change the attributes of a custom object definition without causing any problems because no rules or wizards depend on object definition attributes.

Configuring Object Definition Naming Patterns

You can change the naming pattern of a custom object definition without causing any problems because no rules or wizards depend on object definition naming patterns.

1.2.4 Embedded Objects

An embedded object is a simplified custom object created within another custom object. You cannot define phases, assignees, or rules in embedded objects, so they have few dependencies.

This section provides TeamConnect® Legal Matter Management-specific guidelines regarding adding, deleting, and modifying embedded objects. Performing these procedures is covered by the TeamConnect Enterprise Solution Developer documentation.

Adding Embedded Objects

Adding embedded objects should not cause any unanticipated problems, but you will have to determine how to integrate them into the business solution and modify the object view of the parent object.

Deleting Embedded Objects

If custom blocks depend on the embedded object, deleting the embedded object can cause problems unless you also delete the block. To determine which files use an embedded object, look at the **Blocks** tab of the parent object and analyze the filenames to determine whether any files correspond to the embedded object. If there are any corresponding files, you can delete them accordingly.

Modifying Embedded Objects

You can edit the name of an embedded object without causing problems because CJBs reference the embedded object's unique code, not the object's name.

If custom blocks depend on the embedded object, modifying the embedded object can cause problems. Look at the **Blocks** tab of the parent object and analyze the filenames to determine whether any files correspond to the embedded object. If there are any corresponding files, you can modify them accordingly.

Adding, deleting, and modifying embedded objects' categories and custom fields is addressed in Categories and Custom Fields.

Special Embedded Objects

The following embedded objects in TeamConnect® Legal Matter Management use CJBs to display information and will break custom blocks when the object is deleted or their custom fields are modified:

- Corrective Action (of Dispute)
- Lessee (of Transaction)
- Dispute Cost Center
- Transaction Cost Center
- Related Advice & Counsel Matter (of Dispute)
- Related Advice Matter (of Transaction)

1.2.5 Categories

All categories can be added, modified, and deleted through the **Categories** tab of the respective object definition. Many categories are used by other components, so you should proceed with caution when modifying them.

All custom screens are dependent on categories. For more information, see Custom Screens.

This section provides TeamConnect® Legal Matter Management-specific guidelines regarding adding, deleting, renaming, and reordering categories. Performing these procedures is covered by the TeamConnect Enterprise Solution Developer documentation. Additionally, this section provides tips for searching for category dependencies and highlights categories requiring special attention.

Adding Categories

Adding new object categories has no impact on the application and requires nothing more than the work necessary to add new custom fields, object blocks, search views, and rules that may be associated with the new categories.

Deleting Categories

Deleting a category can seriously impact the application if fields, blocks, rules, or wizards depend on it. Dependent fields are essentially deleted. Dependent blocks might display errors unless they are also removed. Dependent rules will not work properly and may cause java error messages to be displayed.

If a category has no dependent components, its deletion will not cause any issues. However, it is best to proceed with extreme caution by determining whether the category is used by other components and make any necessary modifications.

Renaming Categories

You can edit category names without causing problems because rules and CJBs reference the category's tree position, not the category's name. You may have to modify the UI to be consistent with the category name change.

Reordering Categories

Reordering the display order of categories does not cause any issues because the order of the categories is not referenced by XML files, CJBs, rules, or wizards.

Note: *In TeamConnect® Legal Matter Management 3.4, the default order of all categories is zero.*

Searching for Category Dependencies

To determine dependencies on categories, you can search the source code for a variety of text strings. For example, you can search according to the category's tree position to find all the class files that might use that category. Alternatively, you can search for all categories using the following strings:

- **category='Tree_Position'**—This is useful for finding XML blocks that use specific categories. For example, if you search the source code for **category='DISP_EMPL'**, the search results will

point to XML files such as **WzEmploymentDetailsSYS.xml** (the **Employment Details** block in the **New Dispute** wizard's **Details** page). Thus, you would have a list of all the blocks dependent on the Dispute object's Employment category.

- **CAT_**—This string is useful for determining which class files use categories. You can use it to search for all categories used by class files or append the tree position of a specific category. For example, if you search the source code for **CAT_DISP_EMPL**, the search results include **WzDDetailsASYS.java**. Thus, you would know that the Dispute object's Employment category is used by page actions in the **New Dispute** wizard's **Details** page.

You must have access to the source code (.java files, NOT .class files) to be able to search for such text strings.

Special Categories

The tree positions of the following categories are used by many rules and wizards, so deleting them would result in a major impact:

- General Liability/Claim (for Dispute)
- Lawsuit (for Dispute)
- Subpoena (for Dispute)
- Outside Counsel Attorney (for Dispute and Transaction Involved Party)
- Outside Counsel Attorney : Primary (for Dispute and Transaction Involved Party)
- Outside Counsel Firm (for Dispute and Transaction Involved Party)
- Outside Counsel Firm : Primary (for Dispute and Transaction Involved Party)

1.2.6 Phases and Phase Transitions

TeamConnect® Legal Matter Management 3.4 matters come with the following phases that follow a basic process flow and should be acceptable to most customers:

- Intake
- Open
- Closed
- Reopen

In addition, the Advice & Counsel object has the Research Pending category.

A few rules and wizards reference phases and phase transitions; all of them are visible in the UI because the rule names contain the phase and phase transitions.

This section provides Matter Management-specific guidelines regarding adding, deleting, and modifying of phases and phase transitions. Performing these procedures is covered by the other chapters in this documentation.

Adding Phases and Phase Transitions

Adding phases should not cause any unanticipated problems; you will have to determine how to integrate them into the business solution.

Important: *Adding phases to a live system may negatively impact phase histories.*

Deleting Phases and Phase Transitions

If a rule depends on a phase or phase transition, deleting the phase would cause the rule to break and errors to be displayed.

Important: *Deleting phases on a live system may negatively impact phase histories.*

The only dependency on phases in TeamConnect® Legal Matter Management is the rule that ensures that a contact involved in an open matter cannot be deleted. The Dispute and Transaction objects use the **RuleContactInvolvedInOpenMatterQSYS class** file that depends on the **Open** and **Reopen** phases.

Modifying Phases and Phase Transitions

Editing phases should not cause any unanticipated problems, although you would have to determine how to integrate them into the business solution.

Important: *Editing phases on a live system may negatively impact phase histories.*

1.2.7 Assignee Roles

Assignee roles are defined on the **Assignee Roles** tab of the Advice & Counsel, Dispute, and Transaction object definitions. Some assignee roles are used by other components, so you should proceed with caution when modifying them.

This section provides Matter Management-specific guidelines regarding adding, deleting, renaming, and reordering assignee roles. Performing these procedures is covered by other chapters in this documentation. Additionally, this section highlights assignee roles requiring special attention.

Adding Assignee Roles

Adding new assignee roles has no impact other than the work necessary to add new custom fields, blocks, search views, and rules that may be associated with the new role.

Deleting Assignee Roles

Deleting an assignee role can cause a negative impact if rules depend on it. TeamConnect® Legal Matter Management has the following rules for the Dispute and Transaction objects that depend on the **Attorney** assignee role:

- Set Attorney As Main Assignee - Create - SYS
- Set Attorney As Main Assignee - Update - SYS

If you delete all delivered assignee roles, you should turn off the following rules:

- Require At Least One Assignee For Matter - Create - SYS
- Require At Least One Assignee For Matter - Update - SYS
- Track Changes To Main Assignee - SYS

If an assignee role has no dependent components, its deletion will not cause any issues. However, it is best to proceed with caution by determining whether the assignee roles is used by other components and make any necessary modifications.

When possible, it is preferable to change the name and display order while leaving the unique code as-is rather than deleting an assignee role.

Renaming Assignee Roles

You can edit the names of assignee roles without causing problems because rules and CJBs reference the assignee roles' unique code, not the assignee roles' name. You may have to modify the UI to be consistent with the assignee role name change.

Reordering Assignee Roles

Reordering the display order of assignee roles does not cause any issues because the order of the assignee roles is not referenced by XML files, CJBs, rules, or wizards.

1.2.8 Custom Fields

Many of TeamConnect® Legal Matter Management's fields are used by blocks, rules, wizards, and reports. To determine such dependencies, you can search the source code for the field name.

Business Objects reports depend on field names and the **Include in Data Warehouse** setting. See Reports and the Business Objects documentation for more information about report dependencies.

This section provides Matter Management-specific guidelines regarding adding, deleting, and renaming custom fields. Performing these procedures is covered by other chapters in this documentation.

Adding Custom Fields

Adding custom fields does not cause any impact other than adding them to existing object blocks and CJBs if necessary.

Deleting Custom Fields

Deleting custom fields can impact object blocks and CJBs, rules, wizards, reports, and search views. When deleting fields, you must determine such dependencies and fix any affected files, rules, search views and so on. Hiding fields may be preferable to deleting them.

Renaming Custom Fields

Renaming field labels has no impact because the field name is used by other components, not the label. However, you may have to reconcile the new label with other UI terminology.

Note: *If an object block "hard-codes" the label value, edits to the field label will not be reflected in the UI.*

The field name is essentially a unique ID, so changing the name impacts any dependent object blocks and CJBs, rules, wizards, and search views throughout the system as well as in the API and reports.

1.2.9 Lookup Tables

All TeamConnect® Legal Matter Management lookup tables can be viewed and modified on the **Lookup Tables** screen. Using Categories and Lookup Tables provides a reference of TeamConnect® Legal Matter Management's lookup table items as well as the locations of all the fields and class files that use them.

You can find all instances of lookup tables that are used by class files by searching the source code for the **LT_** text string.

Note: *You must have access to the source code (.java files, NOT .class files) to be able to search for such text strings.*

This section provides Matter Management-specific guidelines regarding adding, modifying, reordering, and deleting items in lookup tables. Performing these procedures is covered by other chapters in this documentation. Additionally, this section highlights lookup tables requiring special attention.

Adding Items to Lookup Tables

Adding lookup tables should not cause any impact other than integrating them into the business solution.

Modifying Items in Lookup Tables

If you must modify a particular lookup table, determine which fields use the lookup table by referring to Using Categories and Lookup Tables and then determine which rules, wizards, or reports use those fields.

With the exception of the **Outside Counsel Evaluation** custom lookup table, you can rename lookup table items to suit your implementation without causing problems. However, the tree position of many lookup table items are used by rules and wizards.

Reordering Items in Lookup Tables

All lookup table items have an integer assigned to them that indicates the order in which they are to be displayed with respect to the other items in the same node. Lookup table items with the same integer are displayed alphabetically.

With the exception of the **Outside Counsel Evaluation** custom lookup table, you can re-order lookup table items to suit your implementation without causing problems. However, you must not change the tree position of any lookup table items because many rules and wizards depend on them.

Important: *Do not change the order of the Outside Counsel Evaluation custom lookup table because several rules and wizards depend on the order of its items.*

Deleting Items in Lookup Tables

Deleting a lookup table can cause serious impact if fields or rules depend on it. Dependent fields are essentially deleted. Dependent rules will not work properly and may cause Java error messages to be displayed.

If a lookup table has no dependent components, its deletion will not cause any issues. However, it is best to proceed with caution by determining whether the lookup table is used by other components and make any necessary modifications.

Lookup Tables Requiring Special Caution

The following lookup tables are essential to the corresponding functionality:

- **Budgeting Type**—Used for budgeting functionality.
- **Contact Relation Type**—This system lookup table is used in several CJBs such as the **Outside Counsel**, **Opposing Party**, and **Jurisdiction** blocks.
- **Matter Type**—Used for Invoice functionality. The tree positions must match the applicable matter objects (DISP and TRAN).
- **Matter Specific Budgeting Type**—Used for changing budgeting mode functionality.

You can safely rename the lookup items in these lookup tables, but you may have to reconcile the naming differences within the UI.

1.2.10 Custom Screens

Custom blocks and object views offer flexibility in screen layout and control over user access to record information. Custom fields are the core component of custom screens.

Custom fields can be created and displayed by category only. For all categories and custom fields you add to an object definition, associated access rights are automatically created. Users and user groups must be assigned the appropriate access rights to be able to view the custom screens.

Note: *Some blocks use the* ***TCLegalJavascript.js*** *file to handle the show/hide behavior of some blocks. You can find all the XML files that use it by searching the source code for* ***TCLegalJavascript.js****.*

This section provides Matter Management-specific guidelines regarding adding, modifying, and deleting custom screens.

Adding Custom Screens

You can add custom screens with no impact other than the work necessary to integrate them with the rest of the product, such as UI design.

Modifying Custom Screens

You can perform many modifications to the custom blocks and object views using the **Admin** menu, but some dependencies in class files may not be readily obvious.

Deleting Custom Screens

If a custom field is deleted (or a category is deleted upon which the field depends) but remaining elements of the design require the field, TeamConnect® Legal Matter Management will display an error that shows which custom field is missing. For more information about determining category dependencies, see Categories.

Some custom screens are used only for wizards and use wizard parameters. It is best not to change or remove wizard parameters, because many CJBs, actions, and conditions are dependent on the parameters.

Searching for Categories Used by XML Files

You can find all categories that are used by XML files by searching the source code for the **category=** text string.

Note: *You must have access to the source code (.java files, NOT .class files) to be able to search for such text strings.*

1.2.11 Rules

TeamConnect® Legal Matter Management 3.4 includes the following general sets of business rules that should meet the common needs of most clients:

- Business rules for matter objects
- Business rules for Involved party objects
- Business rules for Contact system object
- Business rules for Invoice system object
- Business rules for matter budgeting

This section provides Matter Management-specific guidelines regarding adding, modifying, activating, deactivating, and reordering rules.

Adding Rules

If you need to incorporate new rules, you should use the configuration tool in the UI. All rules can be created, viewed, and modified on the **Rules** tab of the corresponding object definition.

If you must use Java code to create the necessary rules, see Utility Class Files and Base Classes and review the corresponding files in the code.

Modifying Rules

If you must modify a rule, do not modify it directly. Instead, copy and rename it and make your modifications to the copy. You should deactivate the original rule.

You should be very careful when modifying rules for the Dispute and Transaction Involved Party objects because there are many dependent fields, categories, and other items. Most of these rules use class files.

Activating Rules

You can activate a rule on the **General** tab of the corresponding rule screen.

Before activating rules, consider the affect it will have on users. For example, if you activate the Contact object rule **Only certain users can update contacts of type External Attorney or External Law Firm - SYS**, only users that belong to the Legal Administrator group as their default group will be allowed to update contact records with the **External : Attorney** or **External : Law Firm** categories.

Deleting Rules

You should not delete rules; instead, deactivate them.

Deactivating Rules

You can deactivate a rule on the **General** tab of the corresponding rule's screen.

However, you must not deactivate or delete the following rules:

- **Create Accounts For Matter Budget - SYS**
- **Set Matter Specific Budgeting Type - Update - SYS**
- **Update Dispute Budget - SYS**
- **Update Transaction Budget - SYS**
- **Create Accounts For Vendor Budget - SYS** (for the Dispute and Transaction Involved Party objects)

By default, the Invoice object requires a matter type and matter. To deactivate this feature, you must modify the **New Invoice** wizard and deactivate the following rules:

- **Require Matter When 'MatterType' Set - Create - SYS**
- **Require Matter When 'MatterType' Set - Update - SYS**
- **Set Invoice Project To Line Items - Update - SYS**
- **Set Invoice Project To Line Items - Post - SYS**
- **Set Invoice Project To Line Items - Create - SYS**

You must modify the **New Invoice** wizard. Set the **Default Value** field of the **WzSingleMatterTypePreference** wizard parameter to **No**. On the Invoice object's **Line Items** tab, set the **Show in Concise View** and **Show in Detailed View** lists to **Show** for the **Project** field.

Reordering Rules

Rules of the same type are executed in order from lowest to highest. You can modify the order of most rules. However, the following rules are exceptions:

- **Set Matter Specific Budgeting Type - Update - SYS** Dispute and Transaction rule MUST have a higher order than **Update Dispute Budget - SYS or Update Transaction Budget - SYS**.
- **Set Non-Root Category As Default - Create - SYS** MUST have a higher order than **Matter Must Have Category Besides Root - Create - SYS**
- **Set Non-Root Category As Default - Update - SYS** MUST have a higher order than **Matter Must Have Category Besides Root - Update - SYS**

1.2.12 Reports

Multiple Business Objects universes for ad-hoc reporting and a set of standard Business Objects reports are an optional feature that can be purchased separately. For information on how to install Business Objects, see the *Installation Guide* for Business Objects.

The universes include:

- Standard Reporting universe
- Matter universe
- Finance universe
- Contact Management universe
- Events Universe
- Admin universe
- Document universe

The Business Objects Analytics option includes these reports:

- Top 25 Outside Counsel Spending
- Top 10 Matter by Spending
- Budget Summary
- Invoice Summary
- Spending by Cost Center
- Spending by Matter Category
- Settlement Payout
- Budget/Expenditure Detail
- Matter Count
- Case Progress

Contact Product Management for specific details regarding Business Objects reports.

Caution: *The Business Objects reports depend on many fields. Changing the* ***Include in Data Warehouse*** *setting of custom Fields can impact the creation of reports. For specific field dependencies, see Custom Fields.*

1.2.13 Wizards

Wizards use categories, custom fields, actions, conditions, and CJBs. Their code is tied to the system and is difficult to modify without breaking.

TeamConnect® Legal Matter Management comes with the following wizards:

TeamConnect Legal Matter Management wizards

Object	Wizard
Advice & Counsel	Create a New Advice & Counsel Matter
Dispute	New Dispute
	Create a Subpoena
Invoice	New Invoice
Involved Party for Dispute	New Jurisdiction
	New Opposing Party
Involved Party	New Outside Counsel
Transaction	New Transaction

The **New Outside Counsel** wizard is essentially the same for the Dispute and Transaction objects (some of their lists vary according to matter type).

When the **Create a New Advice & Counsel Matter** wizard is launched from within a Dispute or Transaction record, it creates an embedded Advice & Counsel record in order to display Advice & Counsel information within a custom block on the **Related Advice** tab. The wizard is essentially the same for the Dispute and Transaction objects.

When the **Create a Subpoena** wizard is launched from **Create a new Dispute**, it creates a Dispute record with **Subpoena** category. From the wizard, you can choose to create a subpoena of type: **Order to produce documents** or **Order to appear**. The wizard performs duplicate records checking based on these fields: **Issuing Court**, **Docket Number**, **Type of Subpoena**. If a new subpoena contains the same **Issuing Court**, **Docket Number**, **Type of Subpoena** values as another subpoena, the user will have an option to create a new record or update an existing record.

The **New Outside Counsel** wizard is particularly dependent on wizard parameters and the following Involved Party categories:

- Outside Counsel Firm
- Outside Counsel Firm : Primary
- Outside Counsel Attorney
- Outside Counsel Attorney : Primary

End users will use various wizards to do the following:

- Create a general matter records
- Create matter records with the Advice & Counsel category
- Add involved parties to matter records
- Create Invoice records

For more information about the wizards, see the corresponding Gap Analysis document and the wizard specifications in Maestro.

Java Classes used by Wizards

The following table shows the Java classes used by specific wizard pages:

Java classes used by wizard pages

Wizard name	Page	Class	Description
Create a New Advice & Counsel Matter	General Information	WzACValidateFieldsQSYS	Validate required fields on the **General Information** page.
		WzACValidateFieldsASYS	If there are empty required fields, display the General Information page and the error message "Required field: fieldname cannot be empty."
		WzACCheckWasRelatedASYS	Checks whether the user made a selection on the **Suggestion** page. If so, the wizard proceeds to the **Use Existing AC** page.
		WzACNoSearchResultsQSYS	Checks whether there are any search results based on user input on the **General Information** page. If not, the wizard proceeds to the **Assignees** page.
	Assignees	CreateEmbeddedObject	When invoked from a dispute or transaction record, this page transition rule creates an embedded advice & counsel record in order to

			display advice & counsel information within a custom block on the **Related Advice** tab.
Create a Subpoena	General Information	WzDSetCategorySubpoenalASYS	Sets the **Subpoena** category for a dispute so that related custom fields will be shown.
		WzDValidateSubpoenaFieldsASYS	Validates required fields on the **General Information** page
	Potential Duplicate Records	WzDSetSelectedSubpoenaASYS	If a duplicate existing subpoena is detected, sets the user's selection to create a subpoena or update an existing subpoena (on the **Please make a Selection** page).
	Matter Relations	WzDCreateOrUpdateSubpoenaASYS	Sets whether to create a subpoena or update an existing subpoena after the user clicks **Finish** on the **Relations** page.
New Invoice	General	WzInvoiceValidateFieldsQSYS	Validate fields on the **General** page. If the values of any required fields are not valid, display an error and do not leave the **General** page.
New Jurisdiction	Jurisdiction Selection	WzJValidateContactsQSYS	Validate the contact fields on the **Jurisdiction Selection** page. If either value is not valid, display an error and do not leave the **Jurisdiction Selection** page.
New Outside Counsel (Dispute)	Outside Counsel Attorney Selection	WzDOCValidatePrimaryAttorneyQSYS	Validate the fields on the **Outside Counsel Attorney** Selection page. If either value is not valid, display an error and do not leave the **Outside Counsel Attorney** Selection page.
	Outside Counsel Firm Selection	WzDOCValidatePrimaryFirmQSYS	Validate the fields on the **Outside Counsel Firm Selection** page. If either value is not valid, display an error and do not leave the **Outside Counsel Firm Selection** page.
New Outside	Outside Counsel	WzTOCValidatePrimaryAttorneyQSYS	Validate the fields on the **Outside Counsel Attorney Selection** page. If

Counsel (Transaction)	Attorney Selection		either value is not valid, display an error and do not leave the **Outside Counsel Attorney Selection** page.
	Outside Counsel Firm Selection	WzTOCValidatePrimaryFirmQSYS	Validate the fields on the Outside Counsel Firm Selection page. If either value is not valid, display an error and do not leave the **Outside Counsel Firm Selection** page.
New Dispute	Details	WzDValidateDetailsQSYS	Validate fields on the **Details** page. If the values of any required fields are not valid, display an error and do not leave the **Details** page.
	General	WzDPopulateCategoriesQSYS	Before proceeding to the next page, populate the user's category selections in the dispute record.
		WzDValidateFieldsQSYS	Validate fields on the **General** page. If not valid, do not leave the **General** page.
		WzDHasNoDetailsQSYS	If users select only the **Contracts** or **Intellectual Property** categories (which have no associated **Details** page blocks), the wizard does not display the **Details** page but goes instead to the **Assignees** page.
	Outside Counsel Attorney Selection	WzDValidatePrimaryAttorneyQSYS	Validate that there is only one primary attorney. If there is more than one primary attorney, an error message is displayed and the wizard does not proceed past the **Attorney Selection** page.
	Outside Counsel Firm Selection	WzDIsNoFirmsSelectedAndFirmsListEmptyQSYS	If there are no contacts with the **External : Law Firm** category, the wizard displays the No Firms Found message on the **Outside Counsel Firm Selection** page. When users click **next**, the wizard proceeds directly to the **Budget** page.
		WzDValidatePrimaryFirmQSYS	Validate that there is only one primary firm. If there is more than one

			primary firm, an error message is displayed and the wizard does not proceed past the **Firm Selection** page.
New Transaction	Details	WzTValidateDetailsQSYS	Validate fields on the **Details** page. If the values of any required fields are not valid, display an error and do not leave the **Details** page.
	General	WzTPopulateCategoriesQSYS	Before proceeding to the next page, populate the user's category selections in the transaction record.
		WzTValidateFieldsQSYS	Validate fields on the **General** page. If the values of any required fields are not valid, do not leave the **General** page.
		WzTHasNoDetailsQSYS	If the user selects a transaction category that has no associated **Details** page blocks, the wizard does not display the **Details** page but goes instead to the Assignees page. ***Note:*** *All delivered transaction categories have associated* ***Details*** *page blocks, so this class is only used if there are additional custom categories.*
	Outside Counsel Attorney Selection	WzTValidatePrimaryAttorneyQSYS	Validate that there is only one primary attorney. If there is more than one primary attorney, an error message is displayed and the wizard does not proceed past the **Outside Counsel Attorney Selection** page.
New Transaction	Outside Counsel Firm Selection	WzTValidatePrimaryFirmQSYS	Validate that there is only one primary firm. If there is more than one primary firm, an error message is displayed and the wizard does not proceed past the **Outside Counsel Firm Selection** page.
		WzTIsNoFirmsSelectedAndFirmsListEmptyQSYS	If there are no contacts with the **External : Law Firm** category, the

			wizard displays the No Firms Found message on the **Outside Counsel Firm Selection** page. When users click **next**, the wizard proceeds directly to the **Budget** page.

TeamConnect® Legal Matter Management's wizards also use CJBs. See the wizard sections of the Gap Analysis documents for the names of the CJB files corresponding to each CJB.

1.2.14 Custom Tools

TeamConnect® Legal Matter Management has the following custom tools:

- **Invoice Validation Rule Settings**—Allows you to activate or deactivate invoice validation rules. For more details, see the *CSM Configuration Guide*.
- **Default Settings**—Allows you to set the budgeting mode for the entire system.
- **Map Service of Process—**Allows you to map SOP actions, agents, and targets. For more details, see the *SOP 1.0 Installation and Configuration Guide*.
- **Time Entry Tool**—Allows you to manage timekeepers' entries.

The **Month** field references the **Fiscal Period** custom lookup table, which includes all twelve months. The default fiscal period is **January**.

Important: *The month should be set before any dispute or transaction records are created. Alternatively, the month can be set just before running the Fiscal Year Tool.*

Default Settings

By default, all the following check boxes in the **Budget for** section are selected:

- Outside Counsel Legal Fees
- Outside Counsel Legal Expenses
- Expert or Other Vendors Fees
- Expert or Other Vendor Expenses
- Internal Expenses
- Settlements or Awards
- Internal Time

Each dispute or transaction record will have corresponding accounts automatically created for each item selected in the **Budget for** section.

By default, Do **not budget by fiscal period** is selected in the **Extend budget level by** section. Corresponding fiscal period accounts are created for each dispute or transaction record when the one of the following options are selected:

- Annually

- Annual+Quarter
- Annual+Month

For example, if **Internal Time** is selected in the **Budget for** section and **Annual+Month** is selected in the **Extend budget level by** section, the following corresponding accounts are created for each dispute or transaction record:

- Internal Fees
- Internal Fees : Annual (one for each fiscal year)
- Internal Fees : Monthly (twelve for each fiscal year)

1.2.14.1 Time Entry Tool

The Time Entry Tool comes with default settings that allow you to use it out of the box. You will need to configure rights for Users/User Groups and Contacts to enable rights to perform Time Entry Tool Administrator and User functions.

Important: *You must set a default rate for each timekeeper or errors will occur when using the Time Entry Tool.*

This section describes:

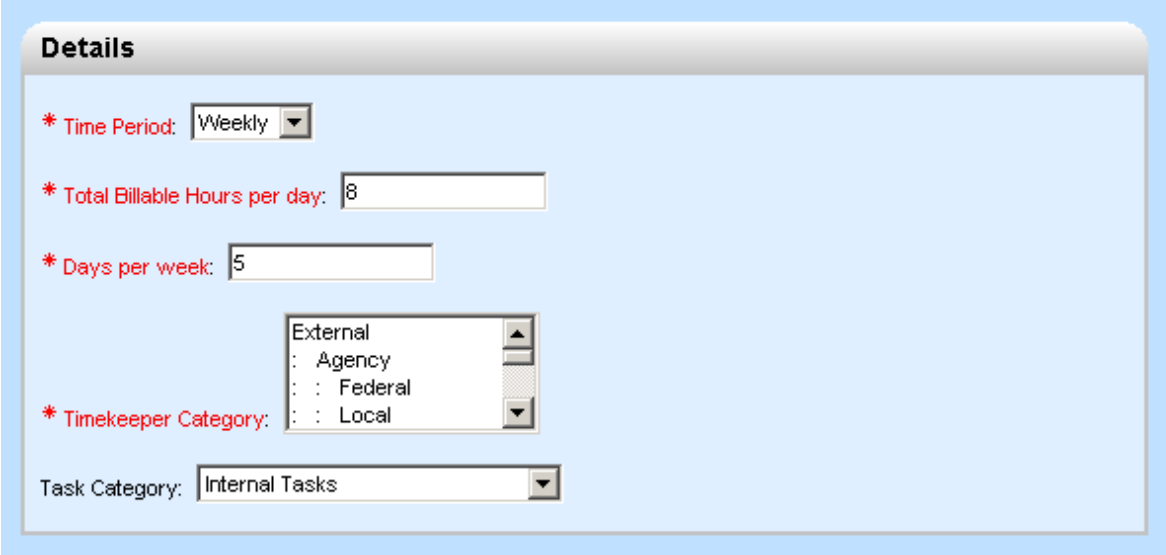
- Out-of-the-box default settings for the Time Entry Tool, as well as how to set up and customize the tool
- Administrator's usage of the Time Entry Tool

For more information, see information about using the Time Entry Tool from a user's point of view.

1.2.14.1.1 Editing Time Entry Tool Settings

You can customize the Time Entry Tool by changing the default settings that determine:

- Billing Period definition
- Daily Billable hours
- Timekeeper required categories
- Task Category



Time Entry Settings Details Block

To edit Time Entry Tool Settings

You can edit the default settings that affect billing Time Period organization, the Time Entry Tool **Available Hours** value, that determine who can use the Time Entry Tool, and available Task Categories from the Time Entry Tool.

1. From the **Main** menu, click the **Go to** drop-down list and select **Time Entry Settings**.
2. Click the **Time Entry Settings** link.
3. Click **Edit**. The following fields are on the **General** tab, **Details** block.
4. (optional) To change the **Time Period**, click the drop-down list and select **Daily**, **Weekly** (default), or **Monthly**. The period determines the time span for which Legal Administrators can define billing Time Periods and time spans for which users can enter time entries.
5. (optional) If you selected **Weekly** for the **Time Period** above, type the billable **Days per week**.

   Days per week only displays if the **Time Period** was set to **Weekly**.
6. (optional) Type the **Total Billable Hours per day** (default - 8). On the Time Entry Tool, this value is used to calculate the total **Available Hours** for a Time Period.

   For weekly time periods, the **Total Billable Hours per day** value is multiplied by the **Days per week** value (described above) to produce the **Available Hours** (displayed on the Time Entry Tool). For example, for one week's time period, the default Available Hours could be (6 Days per week * 8 Total Billable Hours per day) = 48 hours.

   For monthly time periods, Monday through Friday are considered billable (or there are 5 billable days per week). For example, **Available Hours** for the month of June (21 weekdays) would be (21 Days per month * 8 Total Billable Hours per day) = 168 hours.
7. (optional) Select one or more **Timekeeper Categories**. These values determine the default category required for a Matter Management contact to be authorized as a timekeeper (prerequisite to using the Time Entry Tool).

8. To restrict the **Task Category** to describe a new record, click the drop-down list and select a category. In the Time Entry Tool, this value (and child categories) displays as an option from the Category drop-down list when adding time entries.

   In the Time Entry Tool, only tasks (created from Disputes or Transaction Matter records) with the **Default Category** that is specified in the drop-down list as the current **Task Category** will display. For example, if a new task record with **Default Category** of Internal Tasks (default **Task Category** value in Time Entry Settings) is saved, then that task will display as a time entry in the Time Entry Tool.

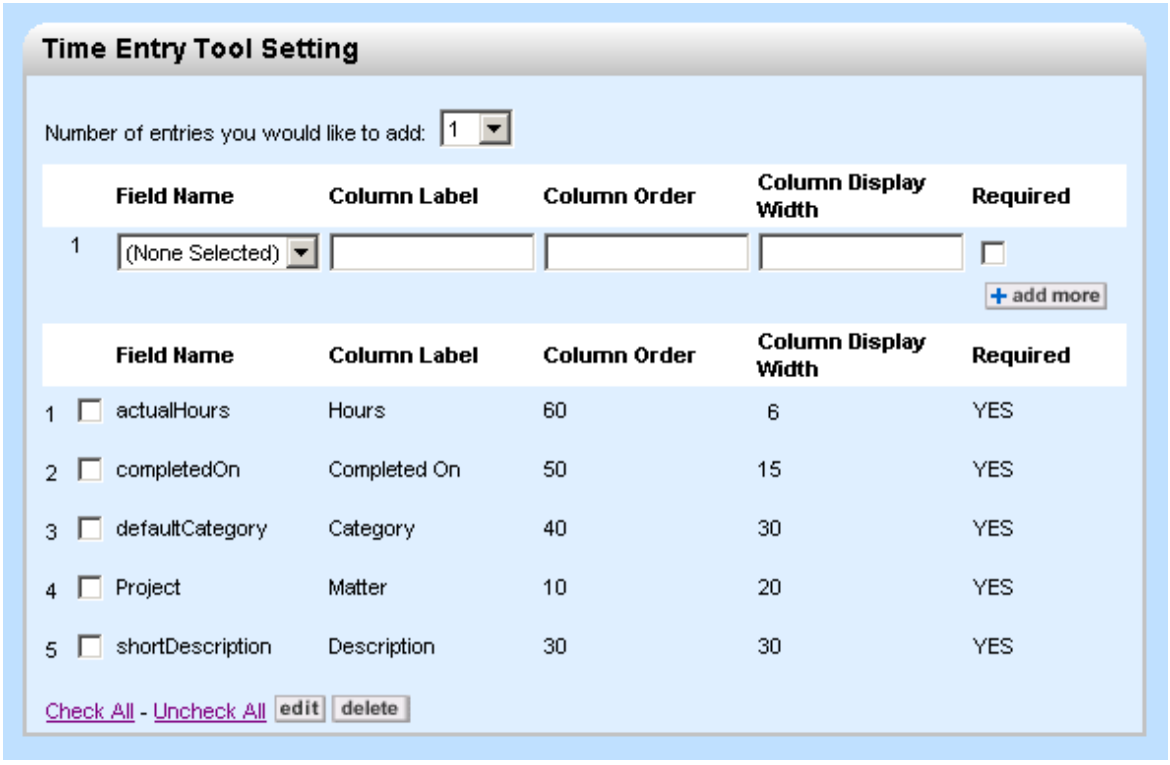
9. Click **Save**.

1.2.14.1.2 Editing Default Time Entry Tool Fields

The Time Entry Tool has been created with default fields and field settings. For all fields, you can change the following settings:

- Column Label
- Column Order
- Column Display Width
- Required

By default, one field, **Task Activity** is hidden. You can activate this field or later hide it again. The existing default fields cannot be hidden.



Time Entry Settings, Time Entry Tool Setting Block

Note: *This image displays default fields and related values for* ***Column Label****,* ***Column Order****,* ***Column Display Width****, and* ***Required*** *fields.*

To edit default Time Entry Tool fields

You can change how Time Entry Tool fields display and whether the fields are required by users.

1. From the **Main** menu, click the **Go to** drop-down list and select **Time Entry Settings**.
2. Click the **Time Entry Settings** link.
3. Click **Edit**. The following fields are on the **General** tab, **Time Entry Tool Setting** Block.
4. To change the text that displays for default fields on the Time Entry Tool:
   a. Check the box(es) next to the existing **Field Name** and click **edit**.
   b. Type the new text to identify the field in the **Column Label** box.
   c. Click **ok** on that field row.
5. To change the position of a Time Entry Tool field from left to right:
   a. Check the box(es) next to the existing **Field Name** and click **edit**.
   b. Type a numeric value in the **Column Order** box, where the lowest value displays furthest to the left and the highest value displays furthest to the right on the screen.
   c. Click **ok** on that field row.
6. To change the maximum length for displaying the field's **Column Label**:
   a. Check the box(es) next to the existing **Field Name** and click **edit**.
   b. Type a numeric value in the **Column Display Width** box. This value determines the length of space allotted to display the corresponding Time Entry Tool field value. For example, for the **actualHours** field, the default **Column Display Width** of 6 would allow a Time Entry Tool user to enter a value like 100.25
   c. Click **ok** on that field row.
7. To set whether an existing Time Entry Tool field is required by users to complete:
   a. Check the box(es) next to the existing **Field Name** and click **edit**.
   b. Clear the **Required** box to display the field but allow blank values, or check the **Required** box prevent saving this field with blank values.
   c. Click **ok** on that field row.
8. To display the **activityItem** field on the Time Entry Tool:
   a. From the **Number of entries you would like to add**: drop-down list, select 1.
   b. From the **Field Name** drop-down list, select **activityItem**.
   c. Type a descriptive field label in **Column Label** box (for example, Task Activity).
   d. Type the field's **Column Order** (number).

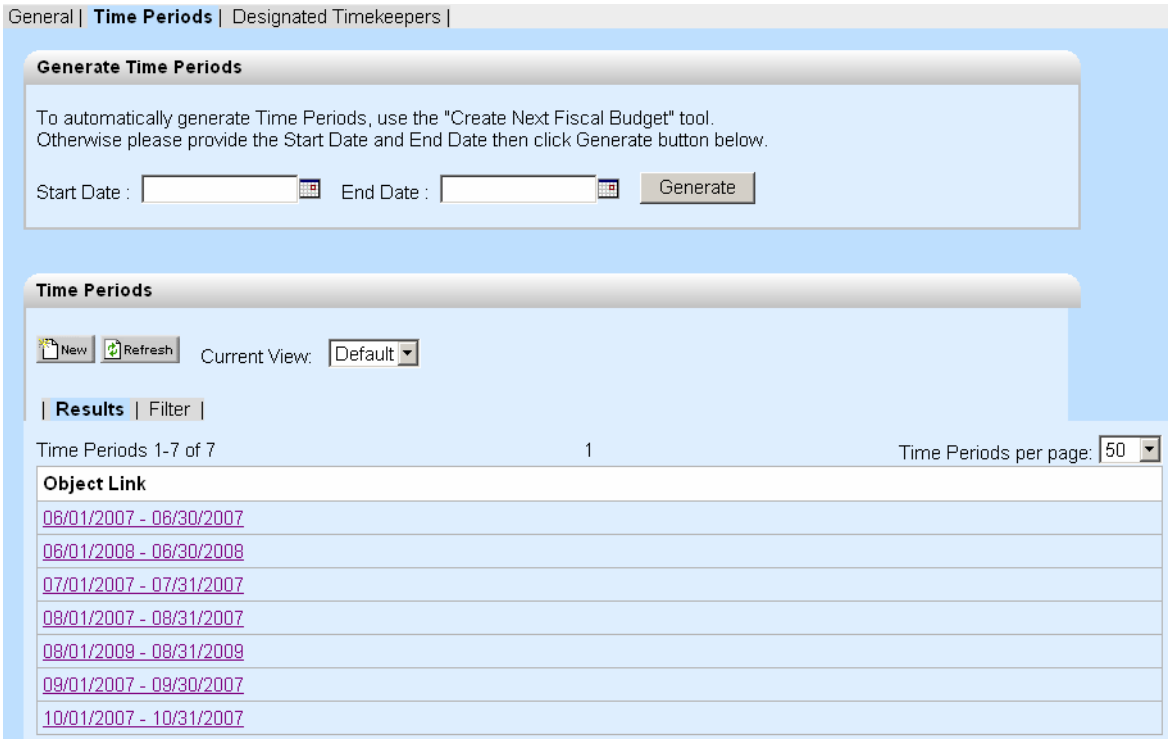
e. Type the **Column Display Width**.

f. Set the **Required** check box. See field explanations above for more information.

9. Click **Save**.

1.2.14.1.3 Creating Billing Time Periods

You can define Time Periods to organize billing periods in which timekeepers enter records for the Time Entry Tool. There are two sections from which you can create a Time Period. You can use the Create Next Fiscal Budget tool (found in the end-user interface in TeamConnect) to create Billing Time Periods that match corresponding budgets (for example, yearly periods), or you can define independent Time Periods from the Time Entry Settings area (for example, on a daily, weekly, or monthly basis).



Time Entry Settigns, Time Periods Tab

You can create Time Periods to match corresponding budgets (using the Fiscal Year tool), or you can create autonomous time periods using the date fields in the Time Entry Settings. For more details, see Default Settings.

To create Time Periods from Time Entry Settings

If you have already created billing Time Periods from the Create Next Fiscal Budget tool, you can skip this procedure. The **Generate Time Periods** section allows quick creation of multiple Time Periods. The **Time Periods** section allows creation of individual Time Periods and overriding the default **Available Hours** calculation.

1. From the **Main** menu, click the **Go to** drop-down list and select **Time Entry Settings**.

2. Click the **Time Entry Settings** link.
3. Click **Edit**.
4. Select the **Time Periods** tab.
5. To create multiple Time Periods quickly, from the **Generate Time Periods** section, click the Calendar icon for **Start Date** and select a date. Click the Calendar icon for **End Date** and select a date.

   If the date range spans multiple time periods, then multiple time periods will be created. For example, if the time period is monthly and the start date is

   06/01/2007 and the end date is 08/31/2007, then three time periods will be created.

   a. Click **Generate**.

   b. Click **Save** (for the Time Entry Setting Record).

      It is recommended to create Time Periods from the Generate Time Periods section because the Time Entry Tool **Available Hours** will be automatically calculated based on the **Billable Hours Per Day** * (billable) **Days in the Time Period**.

6. To create individual Time Periods, from the **Time Periods** section, click **New**.

   a. Click the Calendar icon for **End Date** and select a date.

   b. Click the Calendar icon for **Start Date** and select a date.

   c. Type a number in **Total Billable Hours** (sets the **Available Hours** in the Time Entry Tool).

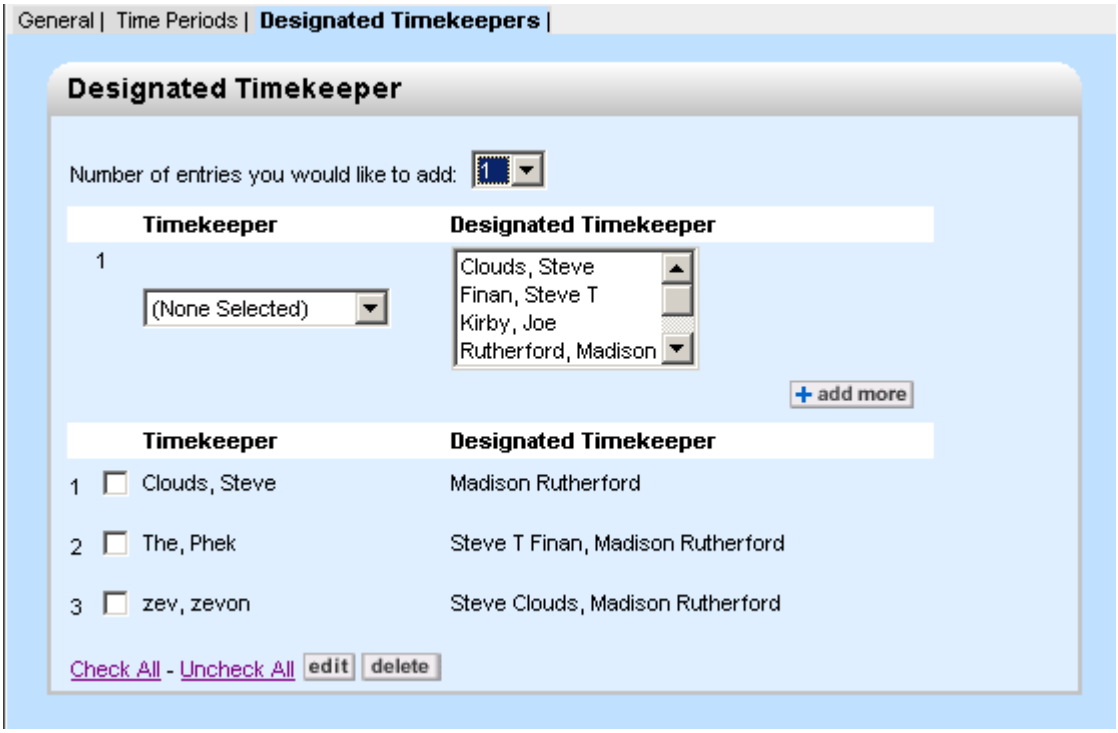
   d. Click **Save**.

Note: *The default naming convention for new Time Periods is "Start Date"-"End Date". For example, 06/01/2007 - 06/30/07.*

1.2.14.1.4 Managing Timekeepers

Members of Legal Administrator group have the authority to set up designated timekeepers. A designated timekeeper has the authority to enter another timekeeper's time entries.

Important: *You must set a default rate in the contact record of each timekeeper or errors will occur when using the Time Entry Tool.*



Time Entry Settings Designated Timekeepers Tab

To delegate Timekeeping duties

Members of the Legal Administrator group can set up designated timekeepers. For example, if Ralph and Tim are timekeepers, Tim can be designated with the ability to enter Ralph's time entries in the Time Entry Tool.

1. From the **Main** menu, click the **Go to** drop-down list and select **Time Entry Settings**.
2. Click the **Time Entry Settings** link.
3. Click **Edit**.
4. Select the **Designated Timekeepers** tab.
5. From the **Number of entries you would like to add**: drop-down list, select the number of timekeepers for which you want to delegate time entry actions.
6. From the **Timekeeper** drop-down list, select the timekeeper to delegate time entry actions for. For example, if you would like Tim to be able to enter Ralph's time entries, select Ralph.
7. From the **Designated Timekeeper** drop-down list, select one or more timekeeper delegates. The maximum number of designated timekeepers for a single Timekeeper is 3.

   For example, if you would like Susan, Tim, and George to be able to enter Ralph's time entries, select Susan. Click the CTRL key and select Tim. Click the CTRL key and select George.
8. Click **ok** for new timekeeper-delegates rows.
9. Click **Save** (for the Time Entry Setting record).

To update the designated Timekeepers list

You can add or remove delegated timekeepers to a Designated Timekeepers list.

1. From the **Main** menu, click the **Go to** drop-down list and select **Time Entry Settings**.
2. Click the **Time Entry Settings** link.
3. Click **Edit**.
4. Select the **Designated Timekeepers** tab.
5. Check the box next to existing **Timekeeper-Delegates** rows to update.
6. Select or deselect **Timekeeper delegates**.
7. Click **ok** for updated **Timekeeper-Delegates** rows.
8. Click **Save** (for the Time Entry Setting record).

To delete the designated Timekeeper-Delegate combinations

You can remove designated timekeeper combinations.

1. From the **Main** menu, click the **Go to** drop-down list and select **Time Entry Settings**.
2. Click the **Time Entry Settings** link.
3. Click **Edit**.
4. Select the **Designated Timekeepers** tab.
5. Check the box next to existing timekeeper-delegates rows to delete.
6. Click **delete**.
7. Click **Save** (for the Time Entry Setting record).

1.2.14.1.5 Using the Time Entry Tool as Administrator

The **Legal Administrator** group can do the following:

- Review and verify the time entries for all timekeepers' entries in the Time Entry Tool.
- Change the phase of a billing time period.
- Add, change (edit, post, void), or delete time entries for other timekeepers.

For more information, see the Legal Matter Management User Help.

Reviewing Time Entries

While reviewing timekeepers' entries in the Time Entry Tool or after the time period has ended, you can lock the corresponding time period. If the time period phase is locked, timekeepers will be prevented from entering, changing, posting, voiding, or deleting time entries from the Time Entry Tool.

To verify time entries for a Time Period

1. From the **Main** menu, **Tools** drop-down list, select **Time Entry Tool**.
2. For the **Task For**, select the time period to view by clicking the Calendar icon to select a date.
   - If the time period is weekly, the corresponding week's time period is displayed.
   - If the time period is monthly, the corresponding month's time period is displayed.
3. From the **Timekeepers** drop-down list, select the timekeeper whose time entries to view.

Time Period Phases

A Time Period can have the following phases:

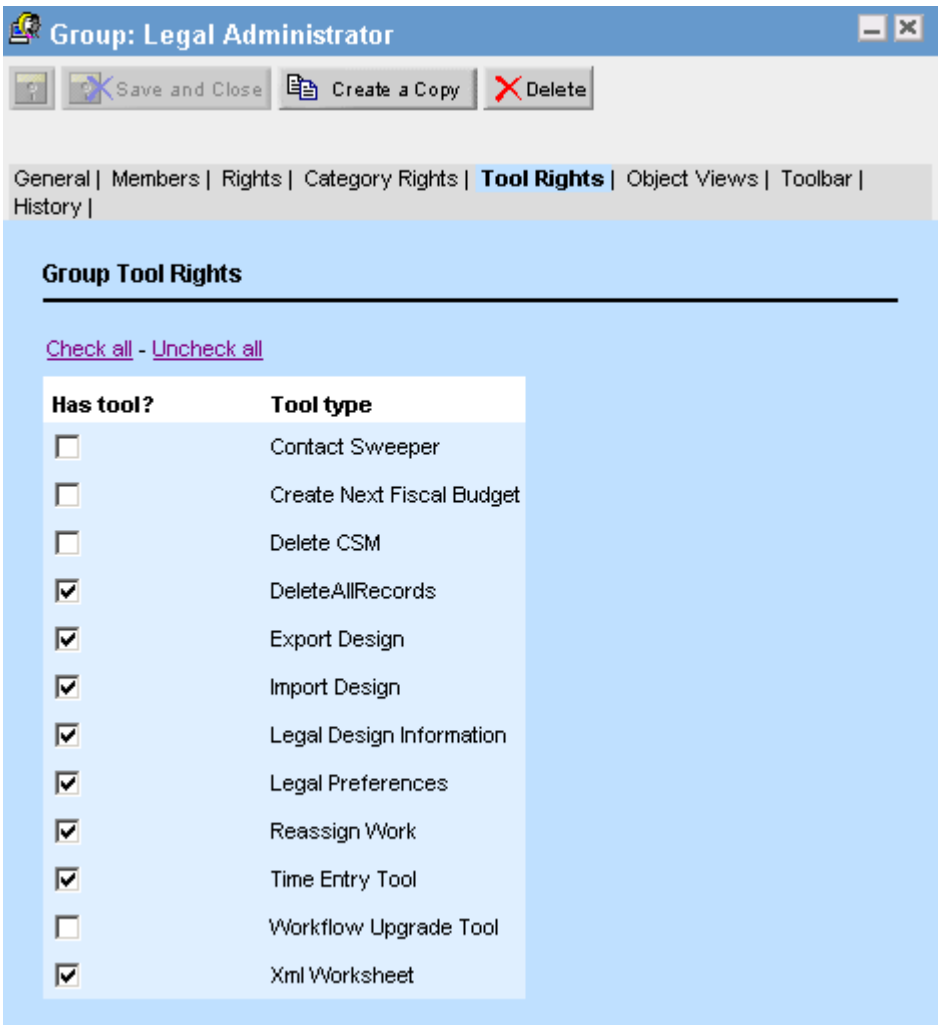
- **Un-Lock—**(default) Allows timekeepers to enter, change (edit, post, void), and delete time entries in the Time Entry Tool
- **Lock—**Prevents timekeepers from entering, changing (editing, posting, voiding), and deleting time entries in the Time Entry Tool. A Time Period in Locked phase can be changed to **Un-Lock** or **Processed** phases.
- **Processed—**Prevents timekeepers from entering, changing (editing, posting, voiding), and deleting time entries in the Time Entry Tool. After a Time Period is set to the **Processed** phase, you cannot change it to previous phases.

To change a Time Period's phase

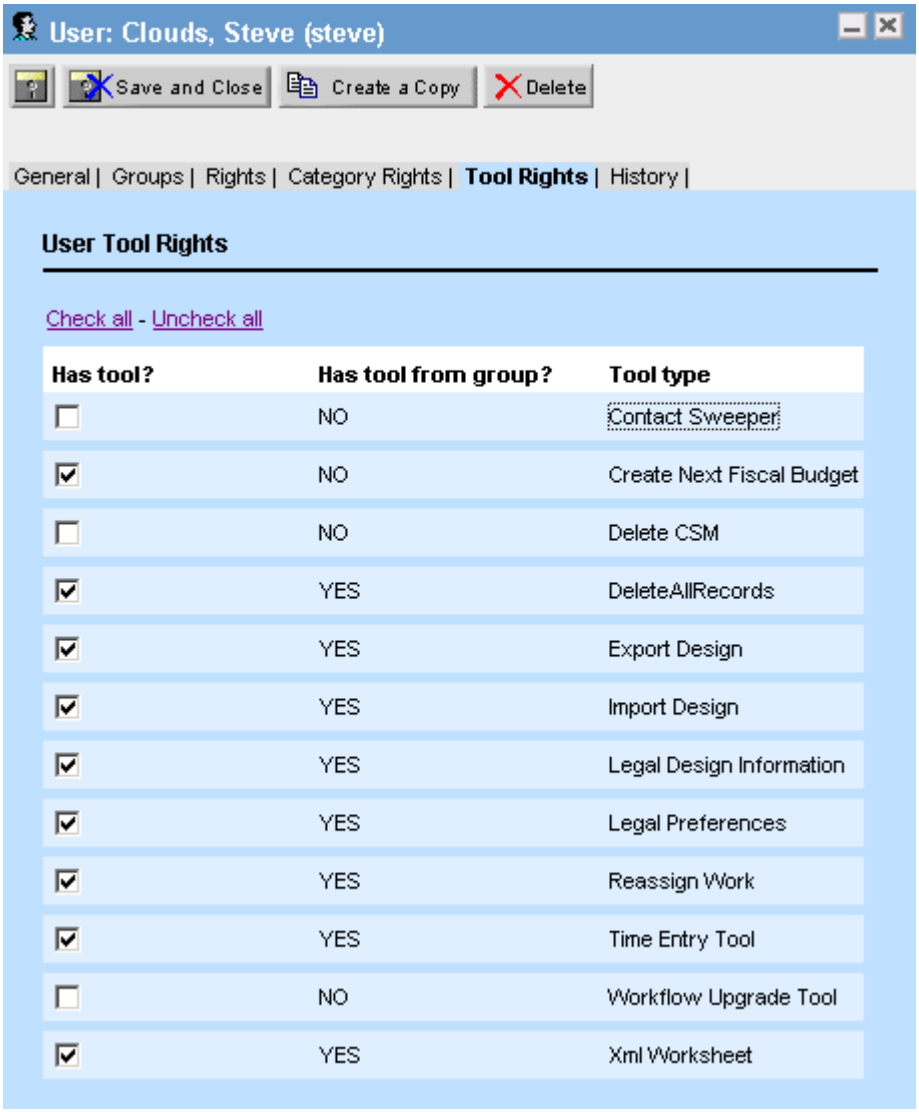
1. From the **Main** menu, click the **Go to** drop-down list and select **Time Entry Settings**.
2. Click the **Time Entry Settings** link.
3. Click **Edit**.
4. Select the **Time Periods** tab.
5. From the **Time Periods** section **Filter** tab, search for a Time Period.
6. From the **Results** tab, click a Time Period name.
7. From the **(actions)** drop-down list, select one of the following:
   - **Un-Lock—**Only displays if the current phase is Lock
   - **Lock—**Only displays if the current phase is Unlocked
   - **Processed**
8. From the confirmation dialog, click **OK**.

1.2.14.2 Tool Rights

You can set the tool rights on the **Tool Rights** tab of the group account screen:



Alternatively, you can set the tool rights according to user on the **Tool Rights** tab of the user's account screen:



1.2.15 CSM Configuration for Matter Management

Three of the Collaborati Spend Management (CSM) Settings tabs are modified for Matter Management:

- E-Billing Roles
- Vendors
- Payments

Information about the default Matter Management configurations is described for reference in this section.

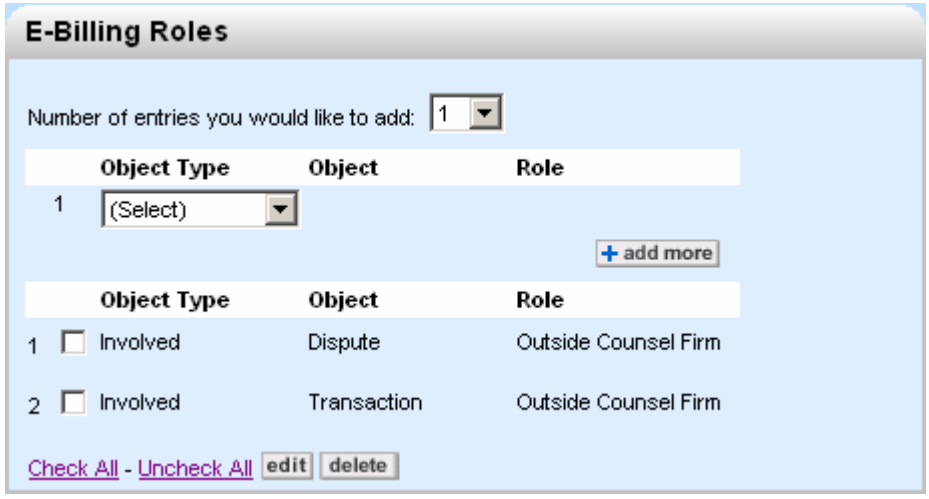
For more information about the default CSM Settings configuration for the **Connection** tab, **Timekeepers** tab, **Default Rates** tab, and **History** tab, see the *Collaborati Spend Management Configuration Guide*.

1.2.15.1 CSM e-Billing Roles Tab Configuration

By default, the CSM Settings, **E-Billing Roles** tab will be configured with the following e-Billing Roles:

E-Billing Roles in CSM Settings

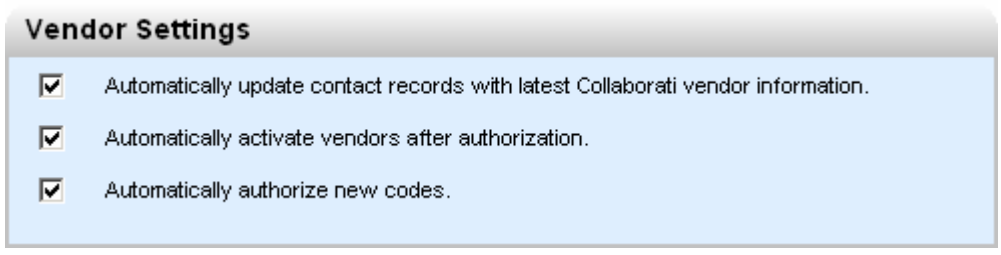
Object Type	Object	Role
Involved	Dispute	Outside Counsel Firm
Involved	Transaction	Outside Counsel Firm



1.2.15.2 CSM Vendors Tab Configuration

By default, the CSM Settings, Vendors tab will be configured with the following Vendor Settings enabled (checked):

- Automatically update contact records with latest Collaborati vendor information.
- Automatically activate vendors after authorization.
- Automatically authorize new codes.



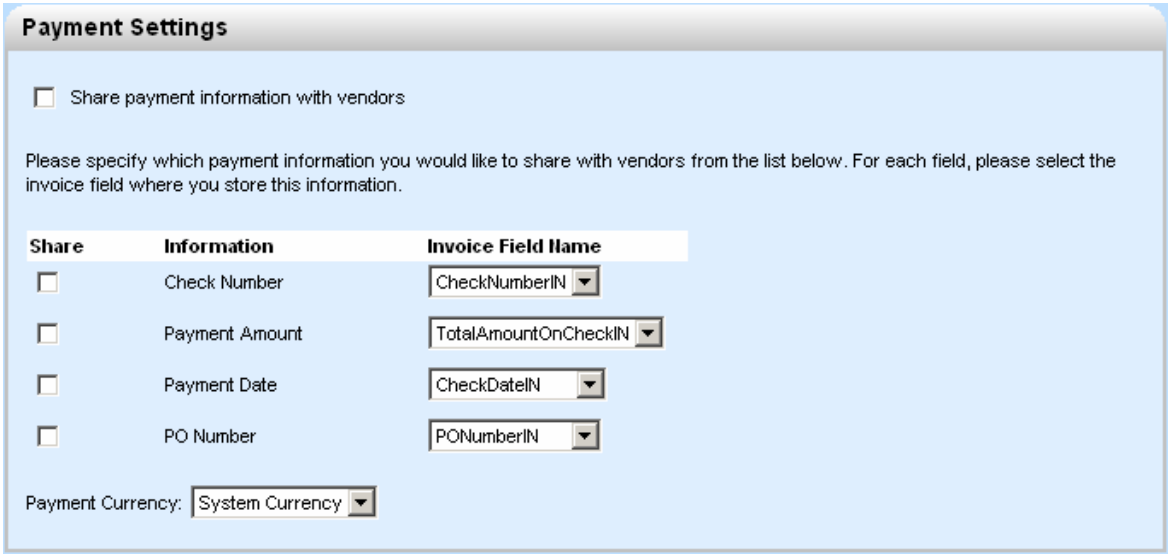
1.2.15.3 CSM Payment Tab Configuration

By default, the **CSM Settings**, **Payments** tab will be configured with the following setting:

Note: *Share payment information with vendors will be disabled (not checked).*

CSM Payment Settings

Share	Information	Invoice Field Name
Not checked	Check Number	CheckNumberIN
Not checked	Payment Amount	TotalAmountOnCheckIN
Not checked	Payment Date	CheckDateIN
Not checked	PO Number	PONumberIN



1.2.16 Groups and Users

TeamConnect® Legal Matter Management 3.4 comes with five user groups with the following sets of pre-configured rights based on their job responsibilities within a legal department:

Pre-configured user group rights

User group	Rights
CSM Admin	Members of the **CSM Admin** group have no Admin, User, or category rights. They have rights only to the following: • **CSM Settings** object and its embedded objects

	• **Activity Code Authorization** object • **Expense Code Authorization** object • **Invoice Payment** object • **Payment Field Mapping** object • **Synchronization History** object • **Task Code Authorization** object • **Timekeeper** object • **Vendor** object and its embedded objects • **Vendor Timekeeper Classification Contact Mgmt** object • **Invoice Validation Rule Setting** tool • **Xml Worksheet** tool
Dispute Attorney	Members of the **Dispute Attorney** group have no Admin rights. They have rights only to the following: • **All User Rights** ***Caution:*** *"All User Rights" includes the Account object.* • **Dispute** object, and its embedded objects • **Advice & Counsel** object • **Cost Center** object and its embedded object • **Category** rights to some **Line Item** categories • **Time Entry Tool** tool ***Note:*** *The contact's default **Category** needs to be set to Internal (or the current Time Entry Setting, **General** tab, **Timekeeper Category** value).*
Legal Administrator	Members of the **Legal Administrator** group have no Admin rights. They have rights only to the Dispute object, its related and embedded objects, and to the following: • **All User Rights** ***Caution:*** *"All User Rights" includes the Account object.* • **Advice & Counsel** object • **Cost Center** object and its embedded object

	• **Dispute** object and its embedded objects • **Time Entry Setting** object and its embedded objects • **Time Period** object • **Transaction** object and its embedded objects • **Category** rights to some **Line Item** categories • **DeleteAllRecords** tool • **Import Design** tool • **Legal Preferences** tool • **Reassign Work** tool • **Time Entry Tool** tool ***Note:*** *The contact's default* ***Category*** *needs to be set to Internal (or the current* ***Time Entry Setting****,* ***General*** *tab,* ***Timekeeper Category*** *value).* • **Xml Worksheet** tool
Paralegal	Members of the **Paralegal** group have no Admin rights. They have rights only to the following: • **All User Rights** ***Caution:*** *"All User Rights" includes the Account object.* • **Dispute** object and its embedded objects • **Advice & Counsel** object • **Cost Center** object and its embedded object • **Transaction** object, and its embedded objects • **Category** rights to some **Line Item** categories • **Time Entry Tool** tool ***Note:*** *The contact's default* ***Category*** *needs to be set to Internal (or the current Time Entry Setting,* ***General*** *tab,* ***Timekeeper Category*** *value).*
Reporting	Members of this group can view the **Reports** home page. This user group has no Admin, User, Category or Tool rights.

SOP Admin	Members of the **SOP Admin** group have no Admin, User, or Category rights. They have rights only to the following:
	• **SOP Settings** object:
	  ○ Project, Read
	  ○ Project, Update
	  ○ Project, All Categories, Read
	  ○ Project, All Categories, Create
	  ○ Project, All Custom Fields, Read
	  ○ Project, All Custom Fields, Update
	• **Retrieval Log** object
	• **Map Service Of Process** tool

SOP Process Manager	Members of the **SOP Process Manager** group have no Admin, Category, or Tool rights. They have rights only to the following:
	• **SOP Settings** object:
	  ○ Project, Read
	  ○ Project, Update
	  ○ Project, All Categories, Read
	  ○ Project, All Categories, Create
	  ○ Project, All Custom Fields, Read
	  ○ Project, All Custom Fields, Update
	• **Consolidated Case Info** object
	• **Retrieval Log** object:
	  ○ Project, Read
	  ○ Project, All Categories, Read
	  ○ Project, All Categories, Create
	  ○ Project, All Custom Fields, Read
	  ○ Project, All Custom Fields, Update
	• **Service** object
	• **Service Of Process - CSC** object
	• **Service Of Process - CT** object
	• **SOP Note** object

	• Contact, Read • Document, Read • Document, All Categories, Read
Transaction Attorney	Members of the **Transaction Attorney** group have no Admin. They have rights only to the following: • **All User Rights** ***Caution:*** *"All User Rights" includes the Account object.* • **Advice & Counsel** object • **Cost Center** object and its embedded object • **Transaction** object, and its embedded objects • **Category** rights to some **Line Item** categories • **Time Entry Tool** tool ***Note:*** *The contact's default* ***Category*** *needs to be set to Internal (or the current Time Entry Setting,* ***General*** *tab,* ***Timekeeper Category*** *value).*

Because users should not alter TeamConnect® Legal Matter Management's automated accounts, you should restrict user access to the Account object. Many rules and wizards depend on the standard configuration.

1.2.17 Service of Process Manager for Matter Management

Service of Process (SOP) Manager for Matter Management supports integration with CT and CSC Registered Agents (RA).

For each Registered Agent, you must add the mapping files to Matter Management in order to use them. Details about adding the mapping files and the contents of the files is described for reference in this section.

For more information about SOP Manager, see the *Service of Process Manager User Guide* and the *Service of Process Manager Installation and Configuration Guide*.

Note: *If you use both RAs, install each mapping file so that all mappings are available to you.*

1.2.17.1 Adding the CT Mapping File to Matter Management

To use SOP mapping, add the CT.xml file to Matter Management. This file is available on your installation media.

To add the CT Mapping file to Matter Management

1. In TeamConnect, go to the following location and create a new document folder named **Mapping Files**:

   Root/System/Object Definitions/Service Of Process - CT

2. On your installation media, in the data* folder, locate the file named **Dispute for CT.xml** and copy it into the *Mapping Files* folder that you created.

1.2.17.1.1 CT Mapping Files

Matter Management is mapped to CT, which enables a Dispute matter to be created based on the mapping files. Participants of the matters are created as Involved(s). The mapping file for CT is named Dispute for CT.

A created Dispute matter contains the following information:

From SOP Field (Type)	To Target Category	To Target Filed	Target Field Label	Target Field Type
Type or name of the lawsuit being served (Text)	DISP	name	Matter Name	Text
Lawsuit reason (Text)	DISP	MatterDescription DI	Matter Description	Text

Default Category = Lawsuit

Note: *Date fields can not be populated when creating dispute matters because the CT source data is in a (Text) format and not a (Date) format as required by the dispute matter.*

The following sections will create a contact record and add it to the client's dataset if a unique identifier is not defined. It will only create a record if a properly formatted contact record is manually created as described in the Service of Process Installation and Configuration Guide.

From SOP Field (Type)	To Target Category	To Target Filed	Target Field Label	Target Field Type
Court Contact (Contact)		.contact	System field	Contact

Case Number (Text)	INPA_COUR	DocketNumberDICO	Docket Number	Text

Involved Role = Court

From SOP Field (Type)	To Target Category	To Target Filed	Target Field Label	Target Field Type
Plaintiff Contact (Contact)		.contact	System field	Contact

Involved Role = Opposing Party

From SOP Field (Type)	To Target Category	To Target Filed	Target Field Label	Target Field Type
Defendant Contact (Contact)		.contact	System field	Contact
Entity Contact (Contact)		.contact	System field	Contact

Involved Role = Defendant

Note: *Only populated SOP records will create Involved(s) in the created Dispute matter. If both SOP records exist, the Dispute matter will contain two defendants*

1.2.17.2 Adding the CSC Mapping File to Matter Management

To use SOP mapping, add the **Dispute for CSC.xml** file to Matter Management. This file is available on your installation media.

To add the CSC Mapping file to Matter Management

1. In TeamConnect, go to the following location and create a new document folder named Mapping Files:

   Root/System/Object Definitions/Service Of Process - CSC

2. On your installation media, in the data* folder, locate the file named **Dispute for CSC.xml** and copy it into the Mapping Files folder that you created.

1.2.17.2.1 CSC Mapping Files

Matter Management is mapped to CSC, which enables a Dispute matter to be created based on the mapping files. Participants of the matters are created as Involved(s). The mapping file for CSC is named **Dispute for CSC**.

A created Dispute matter for CSC contains the following information:

From SOP Field (Type)	To Target Category	To Target Field	Target Field Label	Target Field Type
Name of the transaction (Text)	DISP	name	Matter Name	Text
The date in which the SOP was served (Date)	DISP_LWST	SuitServedDILW	Suit Served	Date
Answer due date (Date)	DISP_LWST	AnswerDueDILW	Answer Due	Date

Default Category = Lawsuit

The following sections will create a contact record and add it to the client's dataset if a unique identifier is not defined. It will only create a record if a properly formatted contact record is manually created as described in the Service of Process Installation and Configuration Guide.

From SOP Field (Type)	To Target Category	To Target Filed	Target Field Label	Target Field Type
Court Contact (Contact)		.contact	System field	Contact
Matter docket number (Text))	INPA_COUR	DocketNumberDICO	Docket Number	Text

Involved Role = Court

From SOP Field (Type)	To Target Category	To Target Filed	Target Field Label	Target Field Type
Plaintiff Contact (Contact)		.contact	System field	Contact

Involved Role = Opposing Party

From SOP Field (Type)	To Target Category	To Target Filed	Target Field Label	Target Field Type
Sender Contact (Contact)		.contact	System field	Contact

Involved Role = Opposing Representative

From SOP Field (Type)	To Target Category	To Target Filed	Target Field Label	Target Field Type
Defendant Contact (Contact)		.contact	System field	Contact
Entity Contact (Contact)		.contact	System field	Contact

Involved Role = Defendant

Note: *Only populated SOP records create Involved(s) in the created Dispute Matter. If both SOP records exist, the Dispute matter contains two defendants.*

1.3 Collaborati Spend Management Customization Help

Welcome to the *TeamConnect® Collaborati Spend Management Customization Help*.

Invoice Validation Rules	Rate Request Approval Rule
AFA Activation and Rules	CSM Settings

1.3.1 Setup Overview

CSM is installed using the Available Updates feature of TeamConnect. Refer to the Installation Help for this version of CSM to learn more about the requirements for installing CSM.

After installation, and prior to configuring CSM, review the prerequisites and setup process to become familiar with the stages of CSM configuration.

1.3.1.1 Prerequisites

The following steps include business and technical prerequisites:

1. Subscribe to CSM. For details, contact your Mitratech account manager. CSM users may need to install Legal Matter Management prior to CSM installation, if desired (see below).

2. Provide your Mitratech account manager with an email address at your organization to be used for notifications related to CSM setup.

3. Provide your Mitratech account manager with a list of vendors to add to e-billing. The list must contain each vendor's country code, default currency, name, Tax Id, address, and a vendor contact person's email address so that vendors can receive email notifications related to Collaborati registration.

   You can add additional vendors at any time by contacting your Mitratech account manager.

4. Create a contact record in TeamConnect for each vendor. Verify that each vendor's contact has been added as an involved party, in active status, to the appropriate matter records.
5. Before running CSM Synchronization, your TeamConnect developers must review any new or existing custom Invoice rules to avoid conflict with CSM behavior.

   For example:

   - If creation of an e-invoice in TeamConnect fails due to a rule triggered on Create, the invoice is rejected. For example, if a validation rule requires an (otherwise optional) invoice field to be populated, an e-invoice might not comply. In this case, the e-invoice will be auto-rejected for the Collaborati vendor with the **Rule Action** error message displayed in the **Comments to Vendor** field.
   - If creation of an e-invoice in TeamConnect fails due to a run-time error in a custom rule, the invoice is rejected. The CSM business admin will receive email notification about the error condition.
   - If an e-invoice has been created in TeamConnect but then fails to post due to a rule triggered on Update or Post, CSM will continue to attempt to auto-post the invoice in TeamConnect until the error is fixed (for example, a rule is fixed or turned off). The CSM business admin will receive email notification about the failure to post.

6. Verify that TeamConnect's task codes, expense codes, activity codes, and non-US tax categories that are used for CSM and Collaborati reflect the values you want your vendors to use in the invoices by doing the following:

   a. Click the **Setup** link from the upper right area of TeamConnect.
   b. Select **Object Definitions** from the **Go To** drop-down list.
   c. Next to the **Invoice** object, click + and then click the **Line Item** object.

      The **Line Item** object definition screen is displayed.

   d. Click the **Task Categories** tab.
   e. Verify that the appropriate task codes are displayed in the **Task Code** column.
   f. Click the **Expense Categories** tab.
   g. Verify that the appropriate expense codes display in the **Expense Code** column.
   h. From the TeamConnect **Admin** menu, select **Lookup Tables** from the **Go To** drop-down list.
   i. On the **System** tab of the **Lookup Tables** screen, select **Activity Items** from the **Show items belonging to** drop-down list.
   j. Verify that the appropriate activity codes are displayed in the **Activity Code** column.
   k. For **Non-US Tax Categories**, return to the **Object Definitions** screen. Click **Invoice**.
   l. Select the **Non-US Tax Categories** tab.
   m. Verify that the appropriate Non-US Tax Categories are displayed in the Tax Code column. Note that **Tax Codes** will vary by the selected Non-US Tax Category node.
   n. If you modify or add any **Task Codes**, **Expense Codes**, or **Activity Codes** use the Line Item Codes Mapping Tool.

7. Verify that CSM administrators (members of the CSM Admin user group who set up your CSM configuration) have sufficient security rights to view the necessary contact records.

   For example, if a vendor's contact record has a security setting of **Private**, the CSM administrator will not have Read rights and cannot establish the necessary relationship between the contact record and the corresponding involved party record.

   In this scenario, you could grant the **Super** user type to the CSM user, which includes Read permissions to private contacts. For details on granting rights, see the TeamConnect Administration Guide.

8. For new TeamConnect installations, determine the appropriate time zone to use for the **Date Received** field of invoice records. This field is time zone independent--it displays the date using TeamConnect's system time zone, not the end-user's time zone. For example, if TeamConnect's **Time Zone** field (on the **Admin Settings / Region** page) is set to **GMT** (the default), the **Date Received** field will display GMT time, even if the user's time zone preferences are set to **EST**.

   For existing TeamConnect installations, it is recommended that you not change TeamConnect's **Time Zone** field on the system settings screen, because this may cause confusion regarding the time zone of original time stamp. That is, if you change the system **Time Zone** setting, the existing data is not updated.

1.3.1.2 Overview of the Initial Setup Process

1. Once you have signed up for CSM, a client record for your law department is created in Collaborati, and your designated contact (as identified in item 2 of Prerequisites) will receive the initial email notification containing a user ID and a password for connecting to Collaborati.

2. Your designated **CSM Admin** group members will need to fill in information on the following pages of the CSM Settings object:
   - **Connection** page
   - **General** page
   - **Default** Rates page
   - **E-billing** Roles page

3. Once the connection has been established, the vendor records for all of your vendors (step 4 of Prerequisites) will be created in Collaborati by Mitratech, and pulled into your CSM settings. This information will enable CSM to pop up suggestions that help a member of your CSM Admin group to correctly map Collaborati vendor records to your vendor contacts.
   - For each authorized vendor, you can optionally restrict the billing codes that they would be able to use (by default, all your invoice line item categories will be available to all vendors).
   - You can deactivate any vendor at any time. Once deactivated, the vendor would not be able to continue submitting invoices electronically.

4. Once the vendor records are mapped and activated in TeamConnect, each vendor will receive a notification inviting them to register for Collaborati.

For each authorized vendor, you may be required to authorize new timekeeper records from Collaborati, mapping them to your existing contact cards (depending on how you have chosen to maintain timekeeper contact cards in TeamConnect). If this is the case, you will be notified once a vendor has registered and submitted timekeepers for your authorization.

5. You can optionally use the **Default Rates** tab to set up a structure to easily specify various levels of rates (based on task codes or timekeeper categories). You can also view expense unit prices.
   - The global-level default settings can be overridden by rates specification at the vendor level or further at the timekeeper level.
   - These rates will be used for invoice validation based on the out-of-the-box e-billing Rules (see Viewing Invoice Validation Rule Settings).
6. You can optionally use the **Payments** tab if your TeamConnect invoices are updated with payment information from your Accounts Payable system, and you would like to share the payment information with your vendors.
7. You can optionally activate any or all of the out-of-the-box e-billing rules that have been created based on most common billing guidelines. These rules provide validation against your established timekeeper rates, expense unit prices, as well as duplicate charges, unauthorized charges (for example, charges for work performed on a closed matter, as established in your guidelines), etc.
8. Budget request information is available in TeamConnect by clicking the **Budgets** link under the **Finance** tab in global navigation. You might also want to add budget request information to other parts of the user interface, such as the pages that display information about your legal matters. Doing so requires a few manual steps:
   a. In **Documents**, navigate to **Top Level / System / Screens**.
   b. Locate block file **BudgetRequestSearchMatter.xml** and copy this file.
   c. Navigate to the **Screens** folder of whatever custom object definition represents your legal matters.
   d. Paste the file into that folder.
   e. In Designer, open that custom object definition and add the block file to one or more object views.

1.3.1.3 What Happens After Setup is Complete

The vendor-specific setting "Also share documents attached to the CSM Settings record with this vendor" is set to "Yes" for newly added vendors. However, you may have some existing vendors where that option is set to "No". If you wish that setting to be "Yes" for all vendors, an upgrade script is available for changing your CSM configuration. It must be run manually, with a SQL query tool, when logged on to the database with the TeamConnect username. The script can be found in the **Documents** page of TeamConnect at this path:

```
Top Level / System / Scripts
```

For MSSQL databases, run script **MSSQL_upgrade_CSM_DocumentSharingOption.sql**

For Oracle databases, run script **Oracle_upgrade_CSM_DocumentSharingOption.sql**

For more information about this setting, see Sharing Documents with a Collaborati Vendor.

First Sync

After completing setup, make sure you synchronize with Collaborati. See Defining CSM Connection and Synchronization for details.

Invoices

Your vendors will immediately be able to start submitting invoices electronically. Once TeamConnect receives submitted invoices, they will be created and submitted for posting automatically, and will follow the same business processes you have established for invoices (for example, invoices will be automatically routed to the desired approval chain based on your approval rules for invoice posting).

Note: *You can optionally create separate workflow rules specific to electronic invoices; use the* ***isElectronic*** *field in the rule qualifier for that option. For details on establishing rule qualifiers, see the Rules Configuration Guide.*

Once invoices are in the approval chain, they display the "Pending Client Approval" status for the respective vendors. Neither intermediate workflow steps (for example, multiple approval steps, ad-hoc reviews, etc.), nor internal correspondence is visible to vendors.

Rejected invoices show the "Rejected" status for the vendors. Approved and posted invoices show the "Approved" status and any adjustments and comments to vendors that were made during the approval process.

Note: *If you selected to share invoice payment information with your vendors, then paid invoices will show the "Paid" status for the vendors. Otherwise, invoice approval and rejection will be the final states communicated to your vendors. If an invoice is voided after posting, this will not be communicated automatically to your vendor but rather, the invoice will remain "'Approved" in Collaborati. If an invoice is re-posted after rejection, this is not communicated automatically to your vendor and the invoice remains in Rejected status in Collaborati.*

New vendors

Repeat steps 3-4 from the initial setup process to authorize new vendors.

To add new vendors to the list, contact your Mitratech account manager with this request.

1.3.1.4 Administer CSM Setup

Each user who requires access to CSM screens needs to be added to the **CSM Admin** group. Do this by following the instructions for maintaining user groups.

1.3.1.5 Viewing Invoice Validation Rule Settings

Submitted invoices go through a comprehensive validation process based on your custom rules (if any) and the active invoice validation rules defined in the Invoice Validation Rule Setting screen.

Note: *There are four invoice types: Standard, Credit Note, Accrual, and Shadow. Invoice validation rules are enforced only against Standard invoices.*

You can deactivate any or all of the out-of-the-box invoice validation rules. Once the rules are activated, they will be active on all invoice updates and newly created invoices (the rules apply to both create and update actions).

Caution: *Do not edit invoice validation rules from the Designer area Invoice object definition. Use the Invoice Validation Rule Setting Tool to disable or enable rules or edit conditional parameters.*

For a complete description of available rules and the values to configure their settings, see Adjusting the Invoice Validation Rule Settings.

Note: *Invoice validation rules only apply to Standard type invoices by default. For further assistance, contact Mitratech Support.*

1.3.1.5.1 Accessing the Invoice Validation Rule Setting Screen

Members of the CSM Admin group have default access rights to the Invoice Validation Rule Setting Tool.

To access the Invoice Validation Rule Setting Screen

1. Click the **Setup** link.
2. From the **Tools** drop-down list, select **Invoice Validation Rule Setting**.

   The **Invoice Validation Rule Setting** screen opens, allowing you to activate or deactivate the out-of-the-box invoice validation rules. For a complete description of available rules and the values to configure their settings, see Adjusting the Invoice Validation Rule Settings.

1.3.1.6 Activating the Rate Request Approval Rule

Rate requests are requests with proposed hourly rates that vendors submit through Collaborati. To be able to accept rates and approve requests, you must activate a rate request approval rule. You also use the approval rule to specify who can approve requests.

Note: *Vendors can only send rates if rate collaboration is enabled. To enable rate collaboration, contact Mitratech Support.*

See Applying Vendor Requested Rates for more information about receiving and approving rate requests.

To activate the rate request approval rule

1. In the Designer, from the **Go to** drop-down menu, select **Object Definitions**.
2. From the **Object Definition List**, navigate to **CSM Settings>Vendor>Rate Request**.
3. Click the **Rules** tab.

4. From the list of rules, click the **CSM - Rate Request Approval** link.
5. Click the **Create a Copy** button, and rename the rule if necessary.
6. Click the **Qualifier** tab. Create rule qualifiers to specify conditions for the rule. See Creating Rule Qualifiers for more information.
7. Click the **Action** tab.
8. Click the **View Route Details** link.
9. Click the **Create a Copy** button, and rename the route if necessary.
10. Create routes to specify who can approve requests. See the sections of Creating Routes for more information.
11. Return to the **Action** tab of the new rule. Select the new route from the **Approval route** field.
12. From the **Process Manager** field, select the group that approvers must be part of to approve rate requests. If you want approvers to be able to approve from **My Approvals**, select the **Workflow Process Manager** group.

    Make sure to add all approvers to the group you select.
13. From **Updates to record pending approval**, select one of the following fields:
    - **Allowed to approvers of current stop**—Allows rate request approvers to update rates of a request.
    - **Allowed to anyone**—Allows any user to update rates of a request.
14. Return to the **General** tab of the new rule, and place a checkmark in **This rule is Active** to activate the rule.

1.3.1.7 Activating Alternative Fee Arrangements and Rules

Instead of using hourly rates to determine an invoice amount, users can enter alternative fees for each vendor on a matter. From the **Alternative Fee Arrangement Settings** screen, you can activate the alternative fee arrangements that appear on the **AFA** page of a matter and determine how they work in CSM.

Note: *Users can only enter alternative fee arrangements if the ability is enabled. To enable alternative fee arrangements, contact Mitratech Support.*

To activate alternative fee arrangements and rules

1. Click the **Setup** link.
2. From the **Tools** drop-down list, select **Alternative Fee Arrangement Settings**. The **Alternative Fee Arrangement Settings** screen opens.

   If you see a message that the AFAs are disabled, but you have enabled AFAs, sync your system with Collaborati. See Starting or Stopping CSM Synchronization with Collaborati for more information.

3. Make alternative fee arrangements active by placing a check-mark in the **This AFA is active** check-box. When you make an alternative fee arrangement active, the option for that alternative fee arrangement appears on the **AFA** page of the matter. Make rules for each arrangement active by placing check-marks in the **This Rule is Active** check-boxes. See the Alternative Fee Arrangements and Rules table for a description of rules.

   If you do not select these check-boxes for active alternative fee arrangements, even if the AFA is active check-box is selected, CSM will not enforce the AFA.

   If conditions for an active rule are met, the following results occur:

   - A default warning message displays in the invoice with an explanation of the condition of the rule. Update the **Default Warning Message** to change the warning message.
   - The invoice automatically adjusts to the alternative fee if you have placed a checkmark in the **Auto-adjust...** check-box of the rule. If the rule is not set to auto-adjust, a message appears in the invoice about updating the invoice manually. See the Alternative Fee Arrangements and Rules table for an explanation of how the invoice automatically adjusts.

     If you use this check-box, select an **Adjustment Reason** so that, if the vendor receives a rejection, the invoice will include information about the reason for the rejection.

4. Select an option for the **Hourly Fee Tracking** field to specify whether or not the invoice tracks the hourly fee details.

   Important: *Make sure you do not change this option after you have already saved the AFA settings. Changing the **Hourly Fee Tracking** field after TeamConnect starts receiving Collaborati invoices can cause issues with the invoice savings amount.*

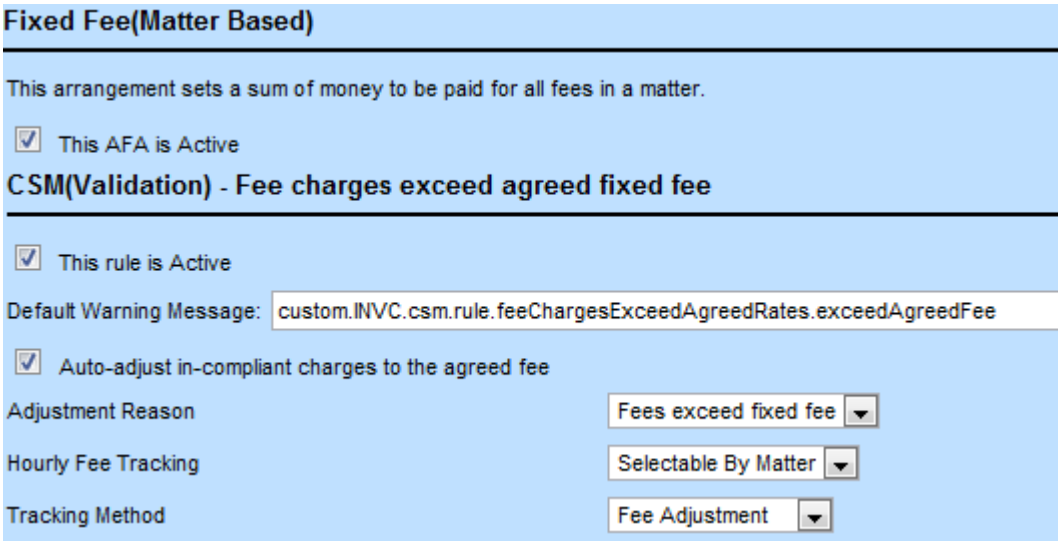
   Volume Discount and **Shared Costs** do not have this option.

   - **Always Disabled—**The invoice does not track hourly fee details.
   - **Always Enabled—**The **Tracking Method** field automatically appears on the matter with the tracking method option that you specify in the setup listed as the default.
   - **Selectable by Matter—**The **Hourly Fee Tracking** check-box appears with the matter so that the user can specify whether or not to track the hourly invoice fees.

5. Select an option for the **Tracking Method** field. The **Tracking Method** for some AFAs is already set.

   - **Fee Adjustment—**A standard invoice includes the original invoice fees with an adjustment to the alternative fee amount.
   - **Shadow Invoicing—**A shadow invoice includes the original invoice fees, while the alternative fee appears as the only amount on the standard invoice.
   - **Selectable by Matter—**The user can specify how to track hourly invoice fees.

6. Click **Save AFA Settings**.



Example of Active Alternative Fee Arrangement and Rule

Details on the available Alternative Fee Arrangements can be found below. For more information on each option, see Adding Alternative Fee Arrangements to a Matter.

Alternative Fee Arrangements and Rules

Alternative Fee Arrangement	Rule	Rule applies if...	When Auto-adjust has a check-mark, the invoice automatically...
Fixed Fee (Matter Based)	**Fee charges exceed agreed fixed fee**	The invoice fee amount exceeds the fixed fee amount.	Updates with the fixed fee amount.
Fixed Fee (Milestone Based)	**Milestone not completed**	No milestones have occurred.	Adjusts to an amount of zero.
	Fees exceed completed milestone fee total	The invoice fee amount exceeds the sum of the completed milestone fees.	Updates with the fixed fee amount.
Fixed Fee (Time Based)	**Invoice Date prior to agreed payment date**	The date on each invoice is earlier than the due date for that invoice. The due date for each invoice is the start date of the next interval	Adjusts to an amount of zero.

	Invoice fee total does not equal Fixed Fee Payment Amount	The invoice fee amount does not equal the total fixed fee amount.	Updates with the fixed fee payment amount.
Capped Fee	**Fees exceed Capped Fee Amount**	The invoice fee amount exceeds the capped fee amount.	Updates with the capped fee amount.
Blended Hourly Rates	**Line item rate exceeds Blended Hourly Rate**	The fee rate on an invoice exceeds the blended hourly rate.	Updates with the blended hourly rate.
Contingency Fee	**Contingency not achieved**	The contingency has not occurred.	Adjusts to an amount of zero.
	Fees exceed Contingency Fee Amount	The invoice fee amount exceeds the contingency fee.	Updates with the contingency fee amount.
Volume Discount	**Discount percent is less than negotiated discount percent**	The sum of all invoice fee amounts for a matter exceeds the threshold.	Updates with the volume discount percentage if the discount percentage on the invoice is less than the volume discount percentage.
Shared Costs	**Matter percent share for fees is incorrect**	The discount percentage on the invoice is higher than the discount of the shared cost (100% - percent share of matter fees).	Updates with the shared cost of fees.
	Matter percent share for expenses is incorrect	The discount percentage on the invoice is higher than the discount of the shared cost (100% - percent share of matter expenses).	Updates with the shared cost of expenses.

1.3.1.7.1 Updating Additional Rules for Alternative Fee Arrangements

After you enable alternative fee arrangements, you may need to make updates to a few additional rules.

If you want TeamConnect to Reject Multi- Matter Invoices

When you initially enable AFAs, if TeamConnect receives a multi-matter invoice and the **AFA Type** for one of the matters is set to **Volume Discount** or **Shared Costs**, TeamConnect rejects the invoice.

To update TeamConnect to reject all multi-matter invoices

1. From the **Tools** drop-down under the **Setup**, select **Invoice Validation Rule Setting**.
2. From the drop-down, select **Show All Rules**.
3. For the **CSM (Validation) - Do Not Allow Multi-Matter Invoice except Line Item Projects with Specific Project Categories** rule, select **This Rule is Active**.
4. Click **Save Rule Setting**.

If You Have TeamConnect Legal

If you have TeamConnect Legal and you enable AFAs, you must update the **AFA Invoice Posting Rule - Post - SYS** rule.

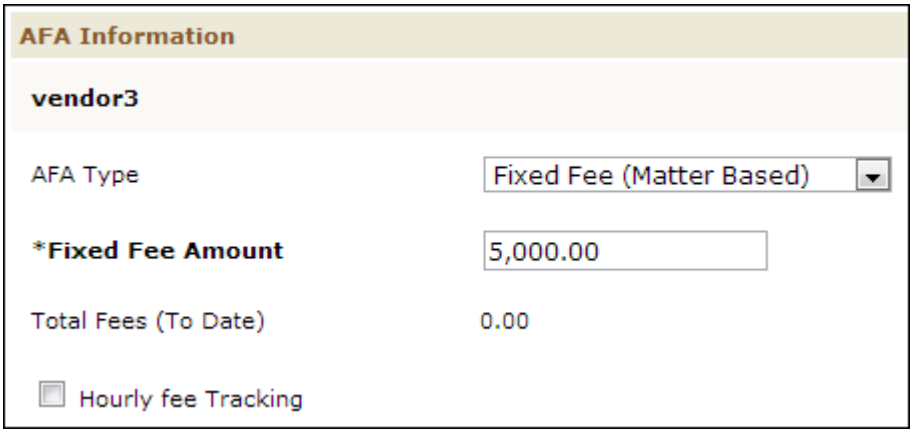
To update TeamConnect Legal when you enable AFAs

1. From the **Go to** drop-down under the **Setup**, select **Object Definitions**.
2. Click the **Invoice** object definition.
3. Click the **Rules** tab.
4. Click the **AFA Invoice Posting Rule - Post - SYS** rule.
5. Change the **Order** to a number higher than 1000.
6. Click **Save and Close**.
7. Log out of TeamConnect to update the changes.

1.3.1.7.2 Creating Alternative Fee Arrangement Pages

After you have activated alternative fee arrangements and their rules, you must configure the **AFA** page to appear.

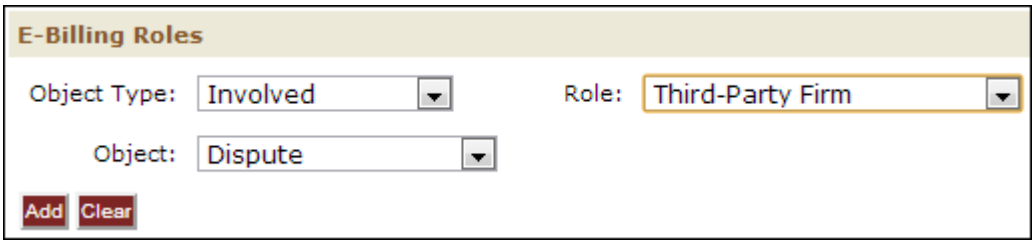
The **AFA** page of a matter or project record is where users enter alternative fee information for each vendor associated with the matter or project.



AFA Page with Fixed Fee (Matter Based) Example

To create the AFA page

1. From the **All** tab, click **CSM Settings**.
2. From **CSM Settings**, click the **E-Billing Roles** link in the left pane.
3. Click **Edit**.
4. Under **E-Billing Roles**, for each type of object record that you want to include the **AFA** page, specify the object if it is not currently specified. Do the following for each object:
   - Select **Involved** from the **Object Type** field.
   - Select the name of the object from the **Object** field.
   - Select the **Role** associated with the object.
   - Click **Add** to add the role to the list.



Specifying that Alternative Fee Arrangements Can Appear for Disputes Example

5. Click the save drop-down, and select **Save & View**.
6. Sync your system with Collaborati. See Starting or Stopping CSM Synchronization with Collaborati for more information.
7. Click the **Setup** link.
8. From the **Go to** drop-down, select **Object Definitions**.
9. Click the object definition link for the matter or project where you want to add the **AFA** page.
10. Click the **Object Views** tab.

11. Click the link for the current object view.

    To look up the current object view, select **Default Object Views** from the **System Settings** drop-down and look for the object name and the chosen object view.

12. In the **Object View** tab that opens, enter **AFA** in the **New** tab name field and click **add**.
13. From the **Add the block** drop-down, select **AFA Information** and click **add** next to the **Add the block** drop-down.
14. Click **Save & Close**.

1.3.2 Uninstalling CSM

Note: *This section assumes that you installed and configured the tool that deletes/restores CSM. For more information, see Creating Custom Tools.*

TeamConnect provides a tool that lets you delete your current CSM design. This is useful if you need to delete an older version of CSM before installing a newer one.

When you use this tool, the following CSM items are deleted:

- CSM settings
- Vendors
- Timekeepers
- CSM object definitions

Non-CSM records, such as Contacts and Invoices, are not deleted if they were created by CSM.

Additionally, all versions of CSM that are listed on the **About/Version** tab are removed and a new entry appears that shows that the delete tool was used to delete the last CSM design.

Note: *While the tool is running, all rules are automatically disabled, and automatically enabled when the uninstall is complete. It is recommended that you perform this task during off-peak hours.*

You can also use this tool to restore CSM default data. See Restoring CSM Default Data for more information.

How Uninstalling CSM Affects your Client Account in Collaborati

When you open the tool, if an active connection exists between CSM and Collaborati, a warning message appears to inform you of the following:

- Uninstalling CSM permanently disables the client account in Collaborati.
- After the uninstall, you must request a new client account if you wish to reinstall CSM.

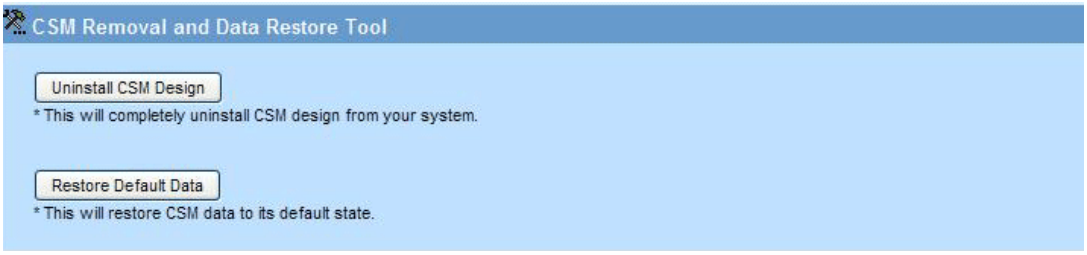
Terminating the corresponding Collaborati client account prevents users from reusing the same client account on Collaborati to synchronize with a different CSM design.

Important: *Before you uninstall CSM, you must turn off synchronization between CSM and Collaborati. See Starting or Stopping CSM Synchronization with Collaborati for more information.*

To uninstall CSM

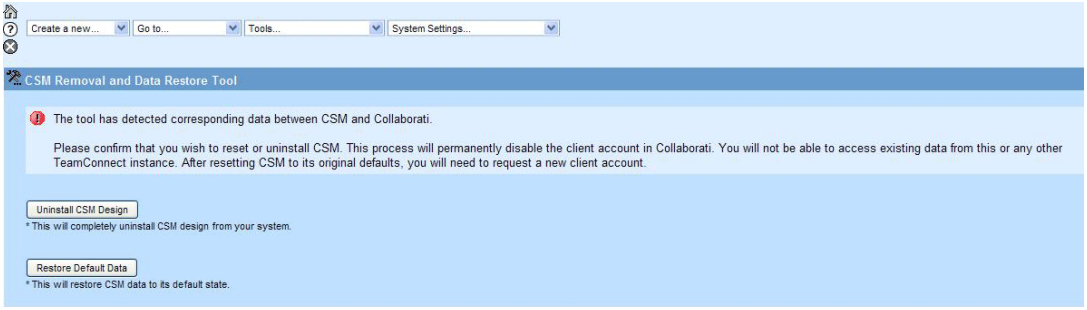
1. In the TeamConnect Designer, click **Tools**, and then select the CSM delete/restore tool.

   The tool's main screen opens.



Delete/Restore Tool Main Screen

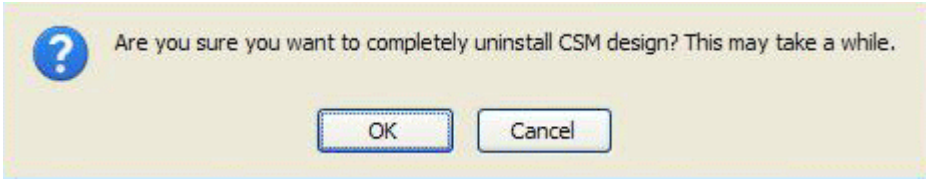
   If the tool detects an active connection between CSM and Collaborati, a warning message appears. This is an informational message and will not prevent the uninstall of CSM.



Warning Message Detecting Active Connection Between CSM and Collaborati

2. To continue, click **Uninstall CSM Design**.

   A confirmation message opens.



Uninstall Confirmation Message

3. Click **OK** to begin uninstalling CSM.

   When the uninstall is complete, a confirmation message appears.

Uninstall Completed Message

4. Click **OK** to close the message.

1.3.2.1 Restoring CSM Default Data

Note: *This section assumes that you installed and configured the tool that deletes/restores CSM default data. For more information, see Creating Custom Tools.*

If you need to drop your CSM settings, but you do not need to upgrade to a later version, use the restore tool to quickly restore default CSM data without removing the CSM design.

Once the default data is restored, you can go to CSM and re-associate vendors. See Authorizing Vendors for more information.

When you use this tool, the following CSM items are deleted:

- CSM settings
- Vendors
- Timekeepers
- CSM object definitions

Non-CSM records, such as Contacts and Invoices, are not deleted if they were created by CSM.

You can also use this tool to uninstall CSM. See Uninstalling CSM for more information.

How Restoring CSM Affects your Client Account in Collaborati

When you open the tool, if an active connection exists between CSM and Collaborati, a warning message appears to inform you about the following:

- Restoring default CSM data permanently disables the client account in Collaborati.
- After you reinstall the default CSM data, you must request a new client account.

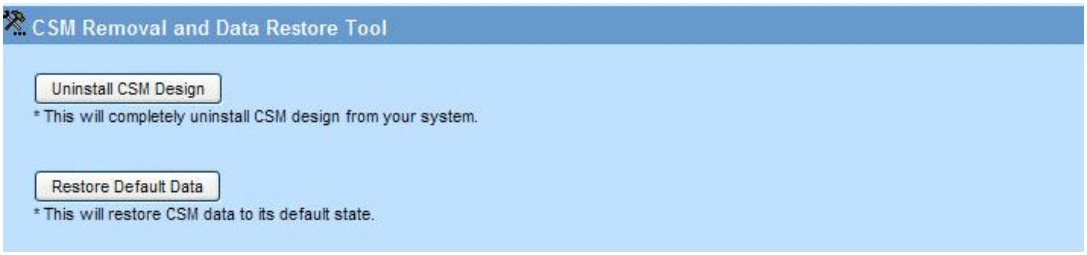
Important: *Before you restore CSM default data, you must turn off synchronization between CSM and Collaborati. See Starting or Stopping CSM Synchronization with Collaborati for more information.*

Note: *While the tool is running, all rules are automatically disabled, and automatically enabled when the uninstall is complete. It is recommended that you perform this task during off-peak hours.*

To restore default CSM data

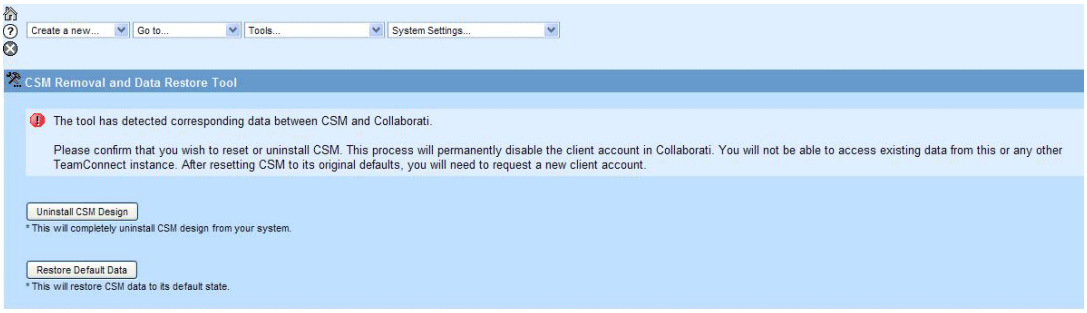
1. In the TeamConnect Designer, click **Tools**, and then select the CSM delete/restore tool.

   The tool's main screen opens.



Delete/Restore Tool Main Screen

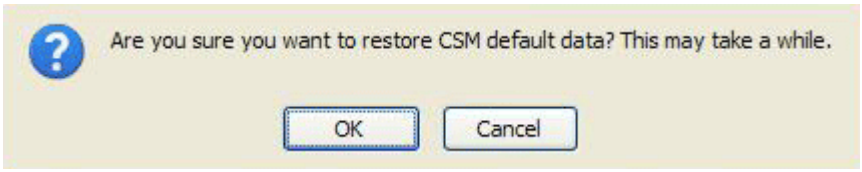
   If the tool detects an active connection between CSM and Collaborati, a warning message appears. This is an informational message and will not prevent the tool from restoring CSM data.



Warning Message Detecting Active Connection Between CSM and Collaborati

2. To continue, click **Restore Default Data**.

   A confirmation message opens.



Restore Confirmation Message

3. Click **OK** to begin restoring CSM's default data.

   When the data is restored, a confirmation message appears.

Restore Completed Message

4. Click **OK** to close the message.

1.3.3 Adjusting the Invoice Validation Rule Settings

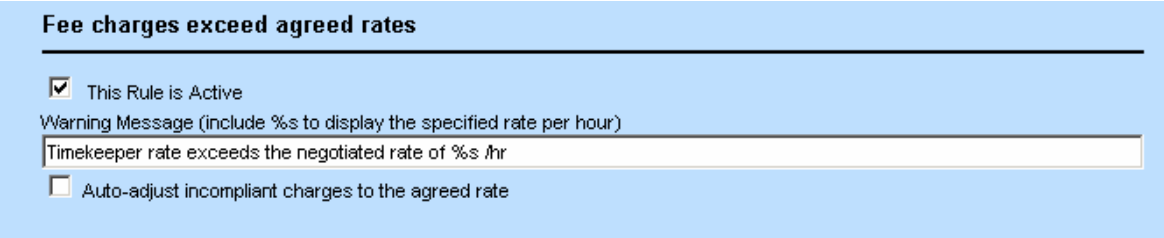
See the following sections for specific rule information:

Caution: *Do not edit invoice validation rules.*

- Fee Charges Exceed Agreed Rates
- Expense Charges Exceed Acceptable Unit Costs
- Do Not Allow Multi-Matter Invoices
- Timekeeper Billed Too Much Time
- Invoice Service/Expense Charges are Too Old
- Invoice is Too Old
- Invoice Contains Duplicate Timekeeper Charge
- Invoice Contains Duplicate Expense
- Invoice Line Item Description Contains Unauthorized Keywords
- Invoice currency doesn't match vendor currency

Note: *This section describes rules defined in the invoice object definition.*

1.3.3.1 Fee Charges Exceed Agreed Rates



Fee Charges Exceed Agreed Rates Rule Settings

Concept:

If rates for timekeepers are set at any level, this rule ensures that all fee charges on an invoice are less than or equal to a timekeeper's set rates for the appropriate time period. This rule will validate that charges do not exceed the "lowest" level rates available for each timekeeper. Rates for timekeepers can be defined at various levels (timekeeper Contact, Vendor, or Global).

Timekeeper rates can be defined at various levels in TeamConnect and CSM. This rule looks for the timekeeper rate in the following order of priority (if the rate is not defined or set to 0, the rule will use the next timekeeper rate definition):

1. Timekeeper Invoice Task Rate (from TeamConnect contact, Rates page) where the rate for a specific line item task code would be used first. If not found, the rate for the parent of a specific line item task code would be used*.
2. Timekeeper Default Rate (from TeamConnect contact, Rates page).

3. Vendor's rate (for associated timekeepers; from CSM Vendor account, Rates page). Depending on settings, the rate for a specific line item task code would be used first. If not found, the rate for the parent of a specific line item task code would be used. Otherwise, the rate for a timekeeper category would be used.
4. CSM Settings Default rate (for timekeepers; from CSM Settings, Default Rates page). Depending on settings, the rate for a specific line item task code would be used first. If not found, the rate for the parent of a specific line item task code would be used. Otherwise, the rate for a timekeeper category would be used.

Note: *For this rule description, the timekeeper's default currency is defined from the TeamConnect contact record, Rates page, Default Currency.*

This validation rule is triggered on invoice creation, and on invoice update.

If an invoice is submitted in a currency that is different from the timekeeper's default currency, this rule will not apply to the invoice.

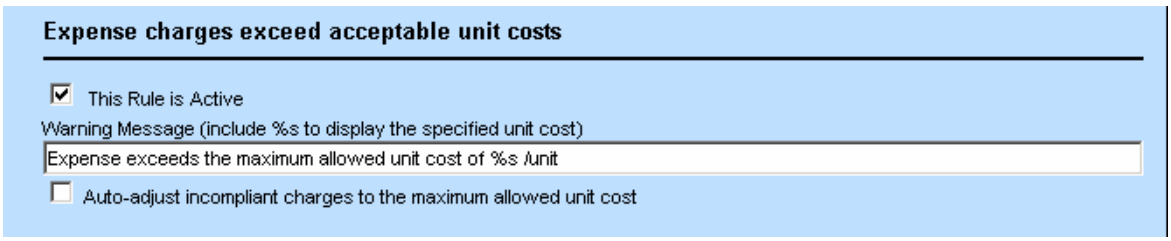
Note: *This is an optional rule; do not set this rule if you do not wish to validate fee charges.*

Field Descriptions:

- **This Rule is Active—**Select this check box to activate the rule. If fee charges exceed agreed rates on invoice line item creation/update, the rule will be triggered. This means that a warning message will be inserted into each incompliant line item charge, unless the **Auto-adjust incompliant charges** option is checked.
- **Warning Message—**Type a warning message to display when this rule is triggered. If the fee charge exceeds the timekeeper's lowest level rate, this warning will be inserted into the line item. The default warning message is: "Timekeeper rate exceeds the negotiated rate of %s /hr", where "%s" is used to display the specified rate per hour.
- **Auto-adjust incompliant charges—**If this option is checked, the fee line item that exceeds your company's specified rate will be auto-adjusted to timekeepers lowest level available rate. The adjustment reason will be defined in the rule parameter "Adjustment reason", if one exists. The **Adjusted By** value will be *System*.

1.3.3.2 Expense Charges Exceed Acceptable Unit Costs

Concept:



Expense Charges Exceed Acceptable Unit Costs Rule Settings

This rule ensures that all expense charges on the invoice are less than or equal to your company's specified unit costs for the appropriate time period. This validation rule is triggered on invoice creation and on invoice update.

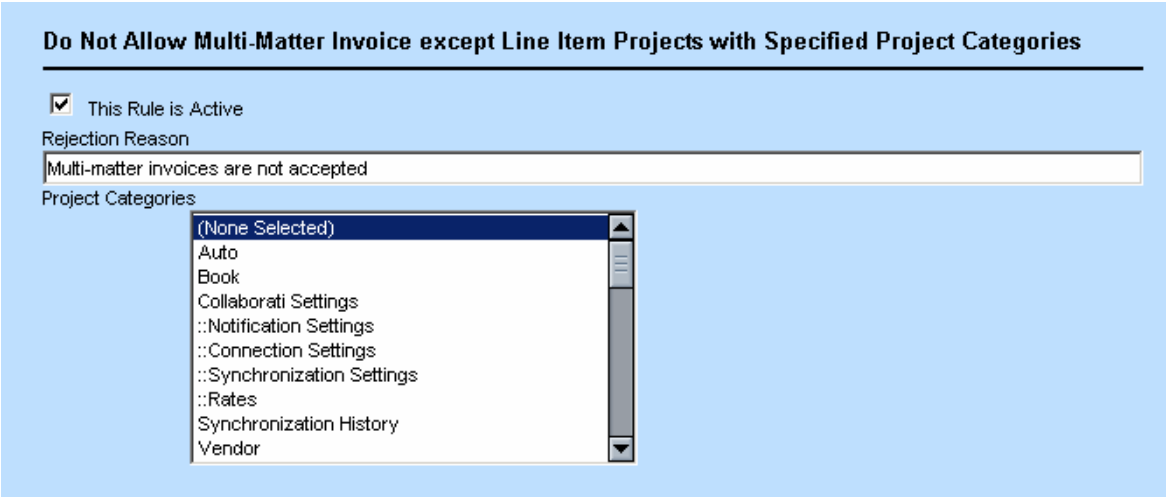
If the submitted invoice's currency is different from the system Default Currency, this rule will not perform any validation on the invoice.

Note: *This is an optional rule; do not set this rule if you do not wish to validate expenses for all vendors.*

Field Descriptions:

- **This Rule is Active—**Select this check box to activate the rule. If expense charges exceed acceptable unit costs on invoice line item creation/update, for each expense line item of the triggered invoice, the rule will be triggered. This means that a warning message will be inserted into each incompliant line item charge, unless the **Auto-adjust incompliant charges to the maximum allowed unit cost** option is checked.
- **Warning Message—**Type a warning message to display when this rule is triggered. If the expense unit price of a line item exceeds your company's specified unit costs (defined on the Invoice Line Item object definition, **Expense Codes** tab), this warning will be inserted into the line item. The default warning message is: "Expense exceeds the maximum allowed unit cost of %s /unit", where "%s" is used to display the specified unit cost.
- **Auto-adjust incompliant charges to the maximum allowed unit cost—**If this option is checked, the expense line item unit price that exceeds your company's specified unit costs will be auto-adjusted to the maximum allowed unit cost. The adjustment reason will be defined in the rule parameter "Adjustment reason", if one exists. The **Adjusted By** value will be *System*.

1.3.3.3 Do Not Allow Multi-Matter Invoices



Do Not Allow Multi-Matter Invoices Rule Settings

Concept:

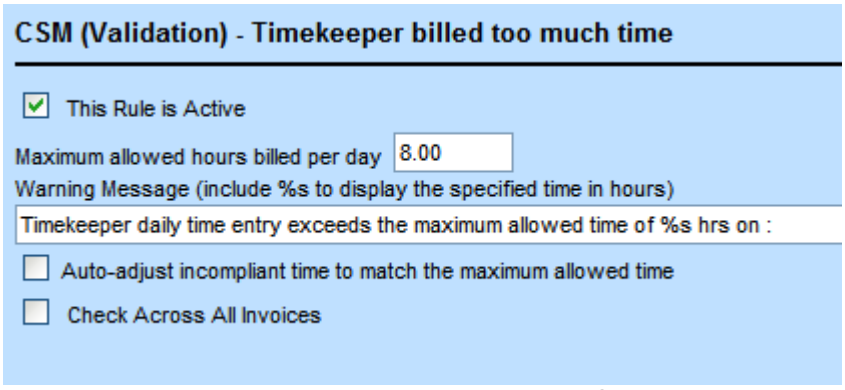
This rule ensures that all invoices submitted for post are either single-matter invoices, or multi-matter invoices that belong to specific project categories for the appropriate time period. This validation rule is triggered on invoice creation (for manual invoices).

Note: *This is an optional rule; do not set this rule if your company allows multi-matter invoices, or if you wish to allow vendors to submit multi-matter invoices under specified conditions.*

Field Descriptions:

- **This Rule is Active—**Select this check box to activate the rule. If an invoice is created with multiple matters, all projects of the invoice will be checked against project categories that accept multiple matters for all line items of the triggered invoice. Incompliant invoices will be automatically rejected.
- **Rejection Reason—**Type a warning message to display when this rule is triggered (manual invoices only). If the triggered invoice has multiple projects and the triggered invoice line item has a project with categories NOT defined in the **Project Categories** parameter, the invoice will be rejected, and this warning will display when the user attempts to save the invoice. The default warning message is: "Multi-matter invoices are not accepted."
- **Project Categories—**Select one or multiple project categories that allow multiple matters to be submitted on an individual invoice.

1.3.3.4 Timekeeper Billed Too Much Time



Timekeeper Billed Too Much Time Rule Settings

Concept:

This rule ensures that all submitted timekeeper time entries on an invoice (or across invoices, depending on your settings) are less than or equal to your company's specified maximum billable hours for timekeepers for the same service date and for the appropriate time period. This validation rule is triggered on invoice creation and on invoice update (for manual invoices).

Note: *This is an optional rule; do not set this rule if your company does not require this type of validation.*

Field Descriptions:

- **This Rule is Active—**Select this check box to activate the rule. The rule is triggered if an invoice is created with timekeeper time entries that exceed the client-specified maximum amount of time for a single service date. This means that a warning message will be inserted into incompliant line item charges, unless the **Auto-adjust incompliant time to match the maximum allowed time** option is checked.
- **Maximum allowed hours billed per day—**Enter the maximum number of hours that can be billed by a timekeeper each day. The default value is 8. This must be a positive, non-zero value.
- **Warning Message—**Type a warning message to display when this rule is triggered. If a line items contains fee charges exceeding the specified time limit, this warning will be inserted into the invoice line item. If you specified the Check Across All Invoices option, all invoices for the same timekeeper and service date will be combined and checked against the limit. The default warning message is:

  "Timekeeper daily time entry exceeds the maximum allowed time of %s hrs.", where "%s" is used to display the specified time in hours.

  For example, Timekeeper daily time entry exceeds the maximum allowed time of 8 hrs on: current invoice (line item 1, 2, 3, ...), INV-101 (line item 1), INV-121 (line item 2,12), INV-129 (line item 20), INV-131 (line item 11),...
- **Auto-adjust incompliant time to match the maximum allowed time—**If this option is checked, the vendor's submitted time that exceeds your company's specified time limit will be auto-adjusted to your company's maximum allowed time. The adjustment reason will be defined in the rule parameter "Adjustment reason", if one exists. The **Adjusted By** value will be *System*.
- **Check Across All Invoices—**Select this check box to enforce this rule on new invoice creation or invoice update against all existing invoice fee line items (for all non-rejected invoices). By default, this check box is cleared.

Sample Scenario One

For example, if the **Check Across All Invoices** check box is selected and the Auto-adjust incompliant time to match the maximum allowed time check box is cleared then the following would apply:

Assume that the invoice validation rule field, Maximum allowed hours billed per day = 8, and an invoice already exists with a fee line item for Timekeeper (John D.), with service date (03/17/2009), and Units (3 hours).

If you create an invoice that contains a fee line item for Timekeeper (John D.), with service date (03/17/2009), and Units (7 hours), the invoice validation rule would be triggered and the sum of billed hours (3 + 7 = 10) would cause the new invoice to be rejected.

Sample Scenario Two

If both **Check Across All Invoices** and Auto-adjust incompliant time to match the maximum allowed time check boxes are selected then the following would apply:

Assume that the invoice validation rule field, Maximum allowed hours billed per day = 8, and an invoice already exists with a fee line item for Timekeeper (John D.), with service date (03/17/2009), and Units (3 hours).

If you create an invoice that contains two fee line items for Timekeeper (John D.), with service date (03/17/2009):

- Fee line item 1: Units value is 2 hours
- Fee line item 2: Units value is 5 hours

Then the invoice validation rule would be triggered and the sum of billed hours (3 + 2 + 5 = 10) would cause the auto-adjust condition to be applied so that the new invoice's Fee line item 2, Unit value would be auto-adjusted to 3 hours. The resulting sum of billed hours (3 + 2 + 3 = 8) would then comply with the invoice validation rule.

The following will occur:

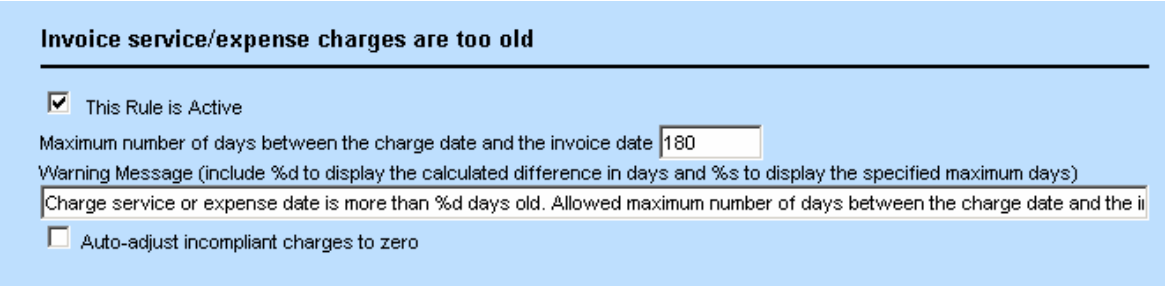
- Under the new invoice's fee line item 2, a warning will display, like Timekeeper daily time entry exceeds the maximum allowed time of 8 hrs on: current invoice (line item 1, 2), INV-101 (line item 1).

  Warnings resulting from breaking this invoice validation rule only display on the line items of invoices you are currently trying to create or update. The warning message will list pre-existing invoice fee line items with the same Timekeeper and service date, that were used to calculate the total hours billed by a Timekeeper.

- The line item detail field for the new invoice's, fee line item 2 will display a Adjusted icon, indicating that the line item was adjusted.

Note: *If you select both the Auto-adjust incompliant time to match the maximum allowed time and* ***Check Across All Invoices*** *check boxes, and if you create/update an invoice that contains fee line items that would break the current conditions of this rule, the auto-adjustment would only apply to the fee line items of a new invoice or invoice you are updating and not be applied to any fee line items of previously created/saved invoices.*

1.3.3.5 Invoice Service/Expense Charges are Too Old



Invoice Service/Expense Charges are Too Old Rule Settings

Concept:

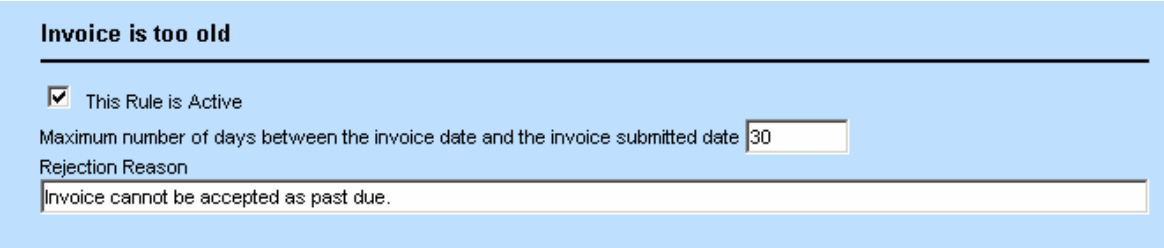
This rule ensures that invoices do not contain service or expense charges (i.e., line items) that are on dates that fall after your company's maximum number of days between the service date and the invoice date. For example, a company may not accept an invoice that contains service dates that are more than 180 days old. This validation rule is triggered on invoice creation and on invoice update (for manual invoices) and settings must be valid within the appropriate time period.

Note: *This is an optional rule; do not set this rule if your company does not require this type of validation.*

Field Descriptions:

- **This Rule is Active—**Select this check box to activate the rule. If an invoice is created with service/expense dates on the invoice (i.e., line item dates) that exceed the client-specified maximum number of days between the service date and the invoice date, the rule will be triggered. This means that a warning message will be inserted into each incompliant line item charge, unless the **Auto-adjust incompliant charges to zero** option is checked.
- **Maximum number of days between the charge date and the invoice date—**Enter the number of days that your company will allow between an invoice date and an individual charge date. The default is 180 days. This must be a positive, non-zero value.
- **Warning Message—**Type a warning message to display when this rule is triggered. If a line item contains service/expense dates exceeding the specified time, this warning will be inserted into the invoice line item. The default warning message is: "Charge service or expense date is more than %d days old. Allowed maximum number of days between the charge date and the invoice date is %s days.", where "%d" is used to display the calculated difference in days and "%s" is used to display the specified maximum days.
- **Auto-adjust incompliant charges to zero—**If this option is checked, incompliant charges will be adjusted to zero. In this case, the information about this adjustment along with the adjustment reason (which should be selected from the Adjustment Reasons lookup table) will be recorded in the adjustment history. The **Adjusted By** value will be *System*.

1.3.3.6 Invoice is Too Old



Invoice is Too Old Rule Settings

Concept:

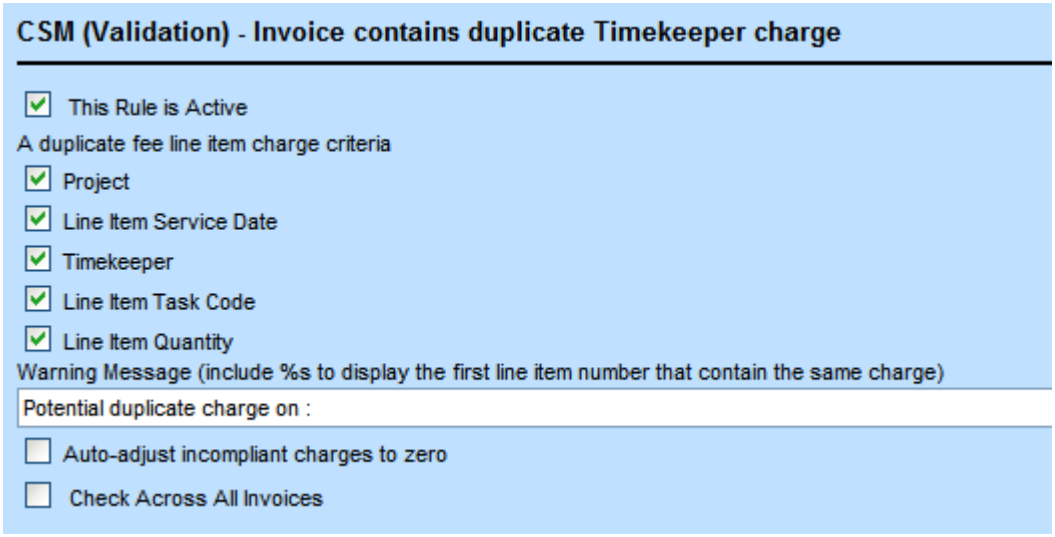
This rule ensures that the calculated difference between the invoice date and the invoice creation date is less than or equal to your company's maximum number of days in which vendors are allowed to submit charges for payment. For example, a company may not accept and invoice with an invoice date that is more than 30 days from the current date. This validation rule is triggered on invoice creation and on invoice update (for manual invoices) and settings must be valid within the appropriate time period.

Note: *This is an optional rule; do not set this rule if your company does not require this type of validation.*

Field Descriptions:

- **This Rule is Active—**Select this check box to activate the rule. If an invoice is submitted with an invoice date that is more than the client-specified maximum number of days between the invoice date and the current date, the rule will be triggered. Incompliant invoices will be automatically rejected.
- **Maximum number of days between the invoice date and the invoice submitted date—** Enter the number of days that your company will allow between an invoice date and the invoice submitted date. The default is 30 days. This must be a positive, non-zero value.
- **Rejection Reason—**Type a warning message to display when this rule is triggered (manual invoices only). If the invoice is submitted after the maximum number of allowed dates, this warning will appear when the user attempts to save the invoice. The default warning message is: "Invoice cannot be accepted as past due."

1.3.3.7 Invoice Contains Duplicate Timekeeper Charge



Invoice Contains Duplicate Timekeeper Charge Rule Settings

Concept:

This rule ensures that vendors do not charge more than once for the same work done on a matter for the appropriate time period. This validation rule is triggered on invoice creation and on invoice update (for manual invoices).

Note: *This is an optional validation rule; do not set this rule if your company does not require this type of validation.*

Field Descriptions:

- **This Rule is Active—**Select this check box to activate the rule. If two fee line items contain a duplicate charge, the rule will be triggered. This means that a warning message will be inserted into each incompliant line item charge, unless the **Auto-adjust incompliant charges to zero** option is checked.

- **A duplicate fee line item charge criteria—**Select the check boxes next to the items that may not have another duplicate fee item charge in the same invoice.
  - Project
  - Line Item Service Date
  - Timekeeper
  - Line Item Task Code
  - Line Item Quantity
- **Warning Message—**Type a warning message to display when this rule is triggered. If the rule is applied to validate only one invoice, a line item contains a duplicate fee item charge, the second and every duplicate line item after will receive this warning. The default warning message is either:
  - "Potential duplicate charge, see line item %s", where "%s" is used to display the duplicated line item number.
  - Potential duplicate charge on: <invoice number or "current invoice"> (line item <line item number>)(line item 1, 2, 3 ...), ...

    For example, Potential duplicate charge on: current invoice (line item 1, 2, 3, ...), INV-101 (line item 1, 3), INV-121 (line item 1, 2), INV-129 (line item 20), INV-131 (line item 11, 14), ...

    In the resulting warning message "..." indicates either more than 3 line items of an invoice or more than 5 invoices break the validation rule.
- **Auto-adjust incompliant charges to zero—**If this option is checked, incompliant charges will be adjusted to zero. In this case, the information about this adjustment along with the adjustment reason (which should be selected from the Adjustment Reasons lookup table) will be recorded in the adjustment history. The **Adjusted By** value will be *System*.
- **Check Across All Invoices—**Select this check box to enforce this rule against all existing invoice fee line items (for non-rejected invoices). By default, this check box is cleared.

Sample Scenario One

For example, if the **Check Across All Invoices** check box is selected and the Auto-adjust incompliant charges to zero check box is cleared then the following would apply:

Assume that all Duplicate Fee Line Item Charge Criteria are selected (Project, Line Item Service Date, Timekeeper, Line Item Task Code, Line Item Quantity). Also assume an invoice already exists with a fee line item for project (Accident Claim 20-304), with service date (03/17/2009), Timekeeper (John D.), Task Code (Accident), and Units/Quantity (3 hours).

If you create an invoice that contains a fee line item for project (Accident Claim 20-304), with service date (03/17/2009), Timekeeper (John D.), Task Code (Accident), and Units/Quantity (3 hours), the invoice validation rule would be triggered and the duplicate values in the Duplicate Fee Line Item Charge Criteria fields would cause the new invoice to be rejected.

A warning would be added to the fee line item of the new invoice.

Sample Scenario Two

If both **Check Across All Invoices** and Auto-adjust incompliant charges to zero check boxes are selected then the following would apply:

Assume that all Duplicate Fee Line Item Charge Criteria are selected (Project, Line Item Service Date, Timekeeper, Line Item Task Code, Line Item Quantity). Also assume an invoice already exists with a fee line item for project (Accident Claim 20-304), with service date (03/17/2009), Timekeeper (John D.), Task Code (Accident), and Units/Quantity (3 hours).

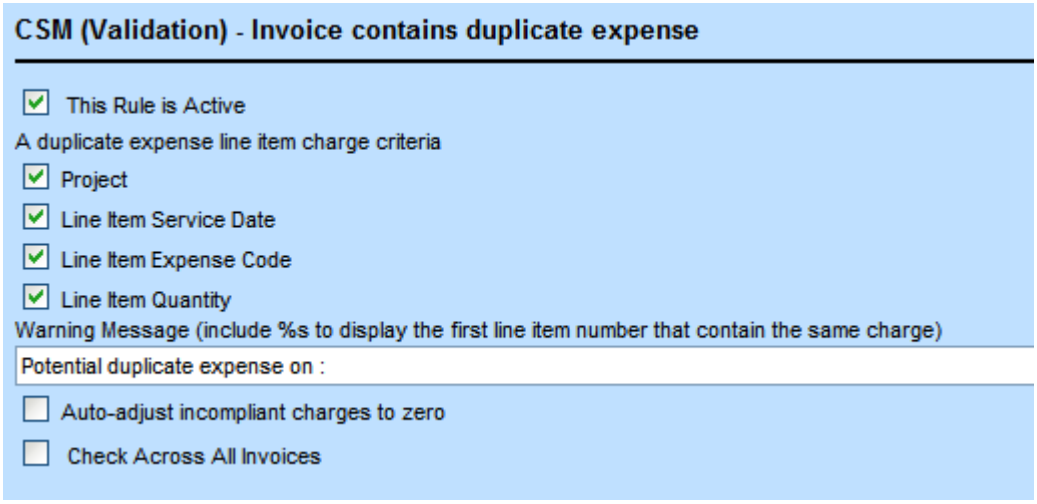
If you create an invoice that contains a fee line item for project (Accident Claim 20-304), with service date (03/17/2009), Timekeeper (John D.), Task Code (Accident), and Units/Quantity (3 hours), the invoice validation rule would be triggered and the new invoice's fee line item Units/Quantity value would be auto-adjusted to 0 hours.

A warning would be added to the fee line item of the new invoice. The Warnings column for the new invoice, fee line item would display a green check mark, indicating that the line item was adjusted.

Note: *Warnings resulting from breaking this invoice validation rule only display on the line items of invoices you are currently trying to create or update. The warning message will list pre-existing invoice fee line items that meet the invoice validation rule's current Duplicate fee line item charge criteria.*

Note: *If you select both the Auto-adjust incompliant charges to zero and* ***Check Across All Invoices*** *check boxes, and if you create (or update) an invoice that contains fee line items that would break the current conditions of this rule, the auto-adjustment would only apply to the fee line items of the new/updated invoice and not be applied to any fee line items of previously saved invoices.*

1.3.3.8 Invoice Contains Duplicate Expense



Invoice Contains Duplicate Expense Rule Settings

Concept:

This rule ensures that vendors do not charge more than once for the same expense on a matter for the appropriate time period. This validation rule is triggered on invoice creation and on invoice update (for manual invoices).

Note: *This is an optional validation rule; do not set this rule if your company does not require this type of validation.*

Field Descriptions:

- **This Rule is Active—**Select this check box to activate the rule. If two expense line items contain a duplicate charge, the rule will be triggered. This means that a warning message will be inserted into each incompliant expense line item, unless the **Auto-adjust incompliant charges to zero** option is checked.
- **A duplicate fee line item charge criteria—**Select the check boxes next to the items that may not have another duplicate expense item charge in the same invoice.
  - Project
  - Line Item Service Date
  - Line Item Expense Code
  - Line Item Quantity
- **Warning Message—**Type a warning message to display when this rule is triggered. If a line item contains a duplicate expense item charge, the second and every duplicate line item after that will receive this warning. The default warning message is either:
  - "Potential duplicate expense, see line item %s", where "%s" is used to display the duplicated line item number.
  - Potential duplicate expense on: <invoice number or "current invoice"> (line item <line item number>)(line item 1, 2, 3 …), …

    For example, Potential duplicate expense on: current invoice (line item 1, 2), INV-101 (line item 1, 3,19, …), INV-121 (line item 2), INV-129 (line item 20), INV-131 (line item 14), …
- **Auto-adjust incompliant charges to zero—**If this option is checked, incompliant charges will be adjusted to zero. In this case, the information about this adjustment along with the adjustment reason (which should be selected from the Adjustment Reasons lookup table) will be recorded in the adjustment history. The **Adjusted By** value will be *System*.
- **Check Across All Invoices—**Select this check box to enforce this rule against all existing invoice expense line items (for non-rejected invoices). By default, this check box is cleared.

Sample Scenario One

For example, if the **Check Across All Invoices** check box is selected and the **Auto-adjust incompliant charges to zero** check box is cleared then the following would apply:

Assume that all Duplicate Expense Line Item Charge Criteria are selected (Project, Line Item Service Date, Line Item Expense Code, Line Item Quantity). Also assume an invoice already exists with a expense line item for project (Accident Claim 20-304), with service date (03/17/2009), Expense Code (Photocopy), and Units/Quantity (50,000).

If you create an invoice that contains an expense line item for project (Accident Claim 20-304), with service date (03/17/2009), Expense Code (Photocopy), and Units/Quantity (50,000), the invoice

validation rule would be triggered and the duplicate values in the Duplicate Expense Line Item Charge Criteria fields would cause the new invoice to be rejected.

A warning would be added to the expense line item of the new invoice.

Sample Scenario Two

If both **Check Across All Invoices** and **Auto-adjust incompliant charges to zero** check boxes are selected then the following would apply:

Assume that all Duplicate Expense Line Item Charge Criteria are selected (Project, Line Item Service Date, Line Item Expense Code, Line Item Quantity). Also assume an invoice already exists with an expense line item for project (Accident Claim 20-304), with service date (03/17/2009), Expense Code (Photocopy), and Units/Quantity (50,000).

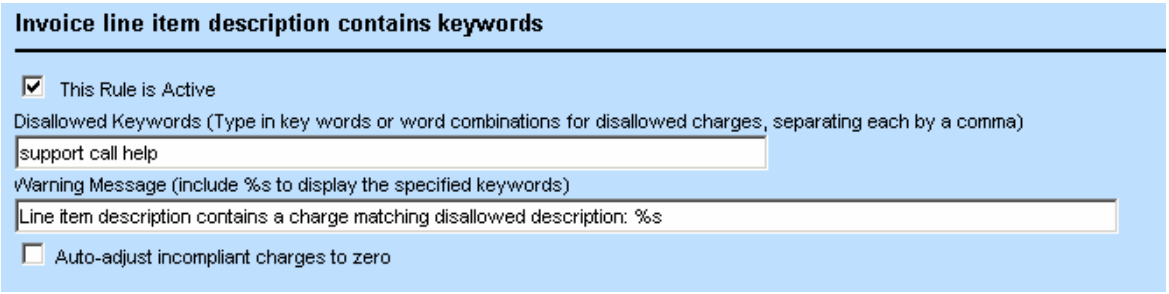
If you create an invoice that contains an expense line item for project (Accident Claim 20-304), with service date (03/17/2009), Expense Code (Photocopy), and Units/Quantity (50,000), the invoice validation rule would be triggered and the new invoice's expense line item Units/Quantity value would be auto-adjusted to 0 units.

A warning would be added to the expense line item of the new invoice. The Warnings column for the new invoice, expense line item would display a green check mark, indicating that the line item was adjusted.

Note: *Warnings resulting from breaking this invoice validation rule only display on the line items of invoices you are currently trying to create or update. The warning message will list pre-existing invoice expense line items that meet the invoice validation rule's current* ***Duplicate expense line item charge criteria****.*

Note: *If you select both the* ***Auto-adjust incompliant charges to zero*** *and* ***Check Across All Invoices*** *check boxes, and if you create (or update) an invoice that contains expense line items that would break the current conditions of this rule, the auto-adjustment would only apply to the expense line items of the new/updated invoice and not be applied to any expense line items of previously saved invoices.*

1.3.3.9 Invoice Line Item Description Contains Unauthorized Keywords



Invoice Line Item Description Contains Unauthorized Keywords Rule Settings

Concept:

This rule ensures that the invoice line item descriptions do not include certain client-specified keywords, for example, "support, help" during the appropriate time period. This validation rule is triggered on invoice creation and on invoice update (for manual invoices). The search for Disallowed Keywords is not case-sensitive and is performed as a wildcard search.

Note: *This is an optional validation rule; do not set this rule if your company does not require this type of validation.*

Field Descriptions:

- **This Rule is Active—**Select this check box to activate the rule. If any of the client-specified keywords are found in the line item descriptions, the rule will be triggered. This means that a warning message will be inserted into each incompliant line item, unless the **Auto-adjust incompliant charges to zero** option is checked.
- **Disallowed Keywords—**Type in any keywords or word combinations that are not allowed in the line item descriptions, separating each by a comma. For example, if you type "one" then the following invoice descriptions would trigger warnings: "ONE", "TONE", "ToNeS", "Phone". Another example, if you would like to get a warning for any line items that reference "library books", then it would be recommended to type "book" rather than "library books" in order to match on all combinations such as "books from library", "a library loaned book", etc.
- **Warning Message—**Type a warning message to display when this rule is triggered. If a line item contains such words, this warning will be inserted into the line item details. The default warning message is: "Line item description contains a charge matching disallowed description: %s", where "%s" is used to display the unauthorized key words.
- **Auto-adjust incompliant charges to zero—**If this option is checked, incompliant line items will be adjusted to zero. In this case, the information about this adjustment along with the adjustment reason (which should be selected from the Adjustment Reasons lookup table) will be recorded in the adjustment history. The **Adjusted By** value will be *System*.

1.3.3.10 Invoice currency doesn't match vendor currency

Concept:

If this rule is enabled and the invoice currency is different from the vendor's default currency (from the TeamConnect contact record, Rates page, Default Currency), the invoice will be rejected. If the vendor's contact default currency is not defined, this rule will compare the invoice currency against the TeamConnect system default currency. This validation rule is triggered on invoice creation and on invoice update (for manual invoices).

You should not enable this rule if you expect your vendors to submit invoices in multiple currencies.

Field Descriptions:

- **Warning Message—**Type a warning message to display when this rule is triggered. If a line item contains such words, this warning will be inserted into the line item details. The default warning message is: "Invoice currency <SUBMITTED CURRENCY CODE> does not match our records for your company: <VENDOR CURRENCY CODE>. Please contact your client's billing coordinator."

- **This Rule is Active—**Select this check box to activate the rule. By default, this rule is not active.

1.3.4 Frequently Asked Questions and Troubleshooting

This section includes additional general information about CSM and tips for troubleshooting CSM.

1.3.4.1 Frequently Asked Questions

How do I enter Non-US Tax Categories?

Detailed instructions for editing Non-US Tax Categories are available in section "Defining Non-US Tax Categories" in the *Customization Guide*.

What kind of email notifications will I receive from Collaborati?

For clients (TeamConnect/CSM customers), the following notifications are sent:

- A welcome email with the user ID and password to connect with Collaborati.
- Email notification when a vendor(s) is added in Collaborati and associated with your (client) company.
- Email notification when a timekeeper(s) is added in Collaborati for a vendor that is associated with your (client) company. Note that if the setting, **Automatically create new Contact records with Collaborati timekeeper information**, is enabled (check) you will not receive this email notification because you will not be required to respond to the timekeeper addition.
- Email notifications will be sent to your company's Business Administrator (email address set on the CSM Settings Connection page) for the following:
    - If creation of an e-invoice in TeamConnect fails due to a run-time error in a custom rule, and the invoice is rejected.
    - If an e-invoice has been created in TeamConnect but then fails to post due to a rule triggered on Update or Post.
- Email notifications will be sent to your company's Technical Administrator (email address set on the CSM Settings Connection page) when your CSM ID (User ID and Password) is changed (on request) by Mitratech Support.

1.3.4.2 Troubleshooting

Invoice Transfers Incomplete on TeamConnect Using WebLogic's JDBC Driver with Oracle

Explanation: If you are running TeamConnect with CSM on a WebLogic application server, using the Oracle JDBC driver that ships with WebLogic, you may notice that no data is synchronizing with Collaborati, even though synchronization shows as running in the CSM Settings screen (for example, expected invoices are not received in TeamConnect). Please check the default system log for a Java ClassCastException on a java.lang.Double.

Resolution: If you find this exception in the logs, we recommend configuring the WebLogic data source using the latest JDBC driver from Oracle.

Unable to start a connection between CSM and Collaborati

Explanation: If you are using an Oracle 10g database for TeamConnect and you are trying to test the connection between CSM Settings and Collaborati, start a connection between CSM Settings and Collaborati, or save an authorized timekeepers list in CSM Settings, if the following error displays:

Connection refused. Unrecognized CSM identifier. The <*encrypted-database-instance-name*>, <*encrypted-database-schema-name*>, <*database-schema-creation-date*>, <*last-csm-connection-date*> do not match the authorized info in Collaborati.

...then you may need to set an additional Oracle JVM parameter.

Resolution: Include the following setting in the JVM argument on the server on which TeamConnect is deployed:

-Doracle.jdbc.V8Compatible=true

For more details, see Oracle MetaLink documentation: Doc ID: Note:360422.1.

Why don't I see CSM Settings or why can't I open CSM Settings?

Explanation: If you have upgraded from TeamConnect 2.x to the current version, you need to do the following to open CSM Settings.

1. Verify that your user is a member of the CSM Admin group. See the TeamConnect Administration Guide for more information.
2. From the TeamConnect User area, from the **All Services** tab, select **CSM Settings**.
3. From the resulting page, you need to click the **All CSM Settings** link from the left navigation pane the first time you open CSM Settings. Afterward, a table will appear in the page body with the **Settings** link available to open CSM Settings.

Why am I seeing warning messages on the CSM Settings Share Documents page or Vendor's Share Documents page?

Explanation: If you are logged in as a user that only belongs to the CSM Admin group and no other groups, you may not have sufficient rights to use the **Share Documents** page. From the **Admin** tab, you need to add your user to a group that has sufficient rights to work with TeamConnect Document records. See your system administrator for assistance.

1.4 Developer's Help

Welcome to the *TeamConnect® Enterprise Developer's Help*.

TeamConnect API Policy	TeamConnect API
Qualifiers and Actions	Custom Pages

Scheduled Actions	Java Samples

1.4.1 TeamConnect API Policy

This API Policy applies when a Client licenses or upgrades to TeamConnect version 4.0 or later. For the purposes of this API Policy, Partners are service providers who successfully complete Mitratech's Partner Certification Program and have signed a partner agreement with Mitratech. Mitratech acknowledges some Clients may engage a third party service provider to perform professional services on Client's behalf; a third party service provider's violation of this API Policy is deemed a violation by Client of this API Policy. Any development performed by Mitratech's Professional Services team will be deemed compliant with this API Policy. The use of the TeamConnect API layer by Mitratech's Clients is subject to the following terms and conditions:

Permitted Use

Interaction with Mitratech products via the API layer is subject to Mitratech licensing rules and agreements. Client acknowledges the API layer contains checkpoints which allow Mitratech to monitor usage of the API layer. Client (including any end users allowed by Client) shall not circumvent the API layer to directly interact with Mitratech products, databases or other components. The end users, whether a named user, limited privilege user, or any other user, of any application accessing the Mitratech API constitute end users of Mitratech products.

Client shall not market or sell an Application that utilizes the API layer without prior written consent of Mitratech and paying the applicable licensing fees, if any.

Policy Violation

If, absent prior written consent of Mitratech obtained via Mitratech's standard consent procedures, Client circumvents the API layer to directly interact with Mitratech products, databases or other components, Client has violated this API Policy. Even if Mitratech consents to the circumvention of the API layer by Client, Client understands the circumventing product will not be warranted or supported by Mitratech, but keeping Mitratech apprised of the situation will help Mitratech better advise Client on future development and upgrades.

Consequences of Violation

Client acknowledges circumvention of the API layer could make future upgrades impractical or cost prohibitive, and Client assumes this risk. In addition, Client understands Mitratech does not support or warrant the specific circumventing development.

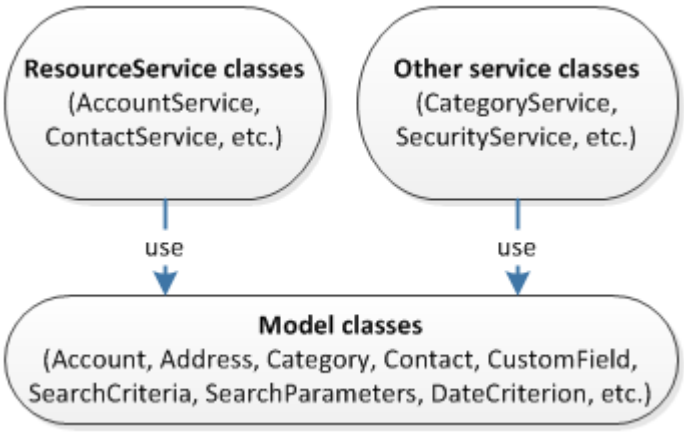
If Client uses a third party service provider, whether or not a Partner, to work on or make any product, database, or component that integrates with TeamConnect, then Client holds Mitratech harmless from any violation of this API policy by the service provider. If Client markets or sells an Application that utilizes the API layer without obtaining Mitratech's prior written consent and paying the applicable licensing fees, if any, Client may be liable for the unpaid license fees and subject to injunction or any other remedies allowed under the license agreement with Mitratech.

1.4.2 The TeamConnect API

If the customization options available in the Designer user interface are not sufficient for your design, you can use the TeamConnect API to add functionality to areas of the application. Written in Java, the TeamConnect API consists of several classes and methods.

Service and Model Classes

The following diagram illustrates the structure of the API.



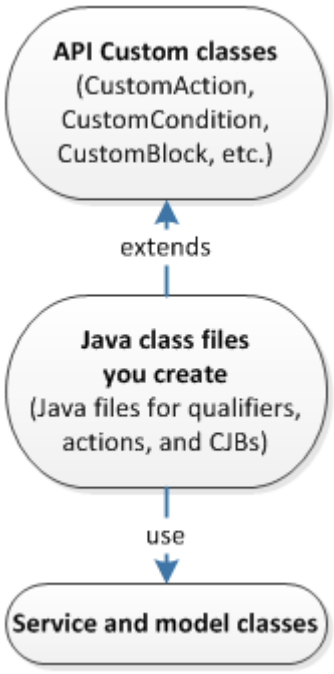
As shown in the diagram, the API includes service classes and model classes. The service classes use the model classes, which contain methods for records and functionality in TeamConnect.

The service classes perform behavior and provide access to instances of an object. The service classes are divided into two different categories:

- ResourceService classes—The service classes that provides methods for creating, reading, updating, deleting, searching for, and manipulating model objects. The API includes a service class for each TeamConnect entity. For example, you use `AccountService` to access financial accounts and `UserService` to access user accounts.
- Other service classes—All other service classes provide methods for high-level functions that do not pertain to one object type. For example, you use `CategoryService` to access category objects and `SecurityService` to access security objects.

Custom Classes

In addition to service and model classes, the API also includes classes for custom code. The following diagram illustrates how custom classes function in the API.



You use the custom classes to create the Java class files for automated qualifiers, automated actions, custom screens, custom tools, and scheduled custom actions. When writing code for these files, you extend the API custom class specific to the type of file you are creating. For example, when creating a Java class for a qualifier, you extend `CustomCondition`. When creating a Java class for a wizard custom screen, you extend `WizardCustomBlock`. After you extend the necessary class, use the service and model class methods to write custom code for the class file.

For example code of each of these types of classes, see Getting Started with Custom Classes.

1.4.2.1 Setting up Your Development Environment

The TeamConnect API is based on JDK 1.6 and works with versions 1.6 and 1.7. You can use any compatible development environment and compiler that you wish. The following JAR file is necessary for custom code development:

teamconnect-enterprise-api-*version*.jar, which contains most TeamConnect packages.

You can find these files in the **utilities/lib** directory of your installation media.

Note: *In these JAR files,* ***version*** *represents the version number that differs from one TeamConnect version to the next.*

To test your custom code, you must have access to an instance of TeamConnect. From TeamConnect, upload your Java classes, XML, or JAR files to TeamConnect's **Documents** area. Refer to Adding Custom Code to TeamConnect for details about adding custom code to TeamConnect.

1.4.2.2 API Package Classes

Written in Java, the TeamConnect API consists of several packages. Packages are methods grouped together with a common theme for organization purposes.

The API includes the following packages:

- **com.mitratech.teamconnect.enterprise.api**—Includes any general classes, mainly `Platform`.
- **com.mitratech.teamconnect.enterprise.api.callable**—Includes the four different types of callable methods that you use with `UtilityService.runAsSystemUser()`.
- **com.mitratech.teamconnect.enterprise.api.custom**—Includes the classes you extend when you are creating custom screens, automated qualifiers, and automated actions.
- **com.mitratech.teamconnect.enterprise.api.model**—Includes the classes for all API objects that are not part of the other model packages.
- **com.mitratech.teamconnect.enterprise.api.model.custom**—Includes the API model classes for all design entities.
- **com.mitratech.teamconnect.enterprise.api.model.enums**—Includes the API model classes for all enumerations.
- **com.mitratech.teamconnect.enterprise.api.model.exceptions**—Includes the API model classes for all exceptions.
- **com.mitratech.teamconnect.enterprise.api.model.parameters**—Includes the API model classes for all parameters.
- **com.mitratech.teamconnect.enterprise.api.model.search**—Includes the API model classes for searching.
- **com.mitratech.teamconnect.enterprise.api.service**—Includes all the service classes.

To see the specific contents of these packages, refer to the API Reference in the following location on your installation media:

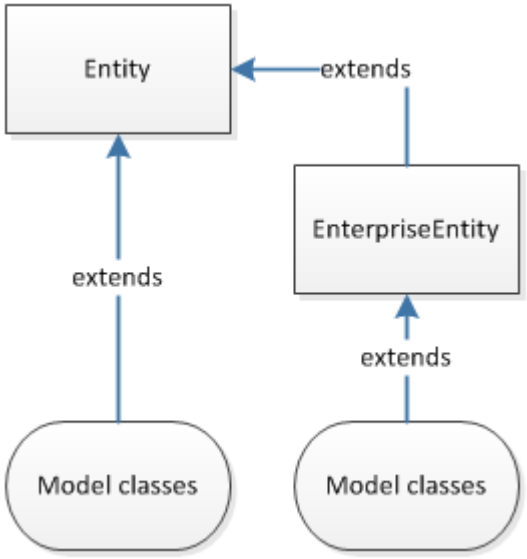
~\utilities\javadocs\api\index.html

1.4.2.2.1 Model Classes

Use the API model classes to describe properties of TeamConnect objects and to implement actions.

Note: *To use model classes, you may need to access a service class if you do not already have access to an instance of the object.*

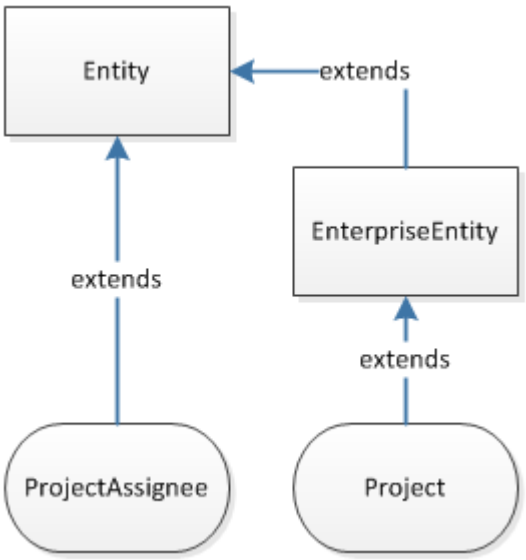
The following diagram illustrates the structure of most API model classes.



All model classes correspond to an object in the database. The TeamConnect API includes the following main model classes:

- `EnterpriseEntity`—The base class for all model classes that correspond to records of system and custom objects. The classes for these objects include methods for record features associated with the objects, such as security, custom fields, and categories. Those model classes extend `EnterpriseEntity`.
- `Entity`—The base class that provides functionality for sub-objects and other properties of `EnterpriseEntity` records, such as assignees, phone numbers, and invoice adjustments. `EnterpriseEntity` extends `Entity`, which means that `Entity` also provides a base class for system and custom objects.

For example, the `Project` model class extends `EnterpriseEntity`, which extends `Entity`. The `ProjectAssignee` model class, which provides functionality for a project, extends `Entity`. The following diagram illustrates the API model structure with these classes.



1.4.2.2.2 Enumerations

Enumerations, or enums, manage options in drop-down lists in TeamConnect. Unlike lookup tables, you cannot add options to these drop-down lists.

All enums are model API classes that include a list of the enum constants. The purpose of these enums is to specify the selected option for a drop-down field, so that you can use the field in API code. Enum classes always include the following two methods:

- `valueOf()`—Returns the enum constant for a given field name.
- `values()`—Returns the list of enum constants.

For example, accounts in TeamConnect have the drop-down list field of **Overdraft Type**. The `AccountOverdraftType` class includes three enum constants for the three options in this drop-down list. You use this class for the **Overdraft Type** account field.

1.4.2.2.3 Service Classes

You access a service class for a particular object when you are using any of the `ResourceService` methods (`create()`, `update()`, `read()`, etc.). For example, if you need to use the `update()` method to update the name of an account, you need to access `AccountService`.

Note: *If you are adding the custom code to a screen or rule, the API already has access to the object with that screen or rule. As a result, you do not need to access the service class for that object.*

When you use the API, you must reference the service classes so that you can call or invoke methods that the service classes use. Accessing the service classes requires two parts:

- A main utility class specific to the module, which you can use to retrieve service implementations. `Platform` is the main utility class for the TeamConnect enterprise module. If you look at the `Platform` class in the javadocs, you see a method that you can use to call each service class for `ResourceService` and `TransationalService`. For example, to access the account service using the `Platform` class, you use the following method:

  ```
  platform.getAccountService()
  ```

- A service class. Service classes provide access to API model classes. To access the account model classes using the main utility class method, you enter the following information:
  - The name of the service class
  - A variable name for the method
  - An equal sign
  - The `Platform` method for the service class

  For example, to access `AccountService`, enter the following code:

  ```
  AccountService accountService = Platform.getAccountService();
  ```

  Note: *Before you can make a change to an object, you must access the service class for the object. If you are using objects from more than one service class in your code, you must access the service classes for both objects.*

After you call the service class(es), you can use the API model classes to describe properties of TeamConnect objects and implement actions.

1.4.2.2.3.1 Resource Services

Resource services provide classes for creating, reading, updating, deleting, searching for, and manipulating model objects. Each service class corresponds to a TeamConnect entity or object type (for example, contacts, projects, accounts, invoices, etc.).

While each service class has it's own set of methods that are specific to that object, `ResourceService` provides a few common functions that all service classes extend, as shown in the following table.

ResourceService Method Actions

Action	Methods to Use
Creating a record	• Use the `create()` method to create a record. ***Note:*** *When writing code to create a record for a rule, condition, or wizard, you do not need to use the create() method because the changes automatically commit to the database.* • Use the `batchCreate()` method to create multiple records.
Updating a record	• Use the `update()` method to update a record. ***Note:*** *When writing code to update a new record for a rule, condition, or wizard, you do not need to use the update() method because the changes automatically commit to the database.* • Use the `batchUpdate()` method to update multiple records.
Reading a record	• Use the `read()` method to read a record using it's primary key. • Use the `readLastSaved()` method to read the last saved version of a record. You can use this version to determine if a record property has any updates.
Deleting a record	***Note:*** *When you want to delete a record that is a dependency for another record, consider the best way to delete that record.* • Use the `delete()` methods to delete a record:

	○ One `delete()` method deletes an object for which you have a reference. ○ The other `delete()` method deletes an object for which you have no reference and the primary key is known. • Use `batchDelete()` to delete multiple records. `batchDelete()` uses the `delete()` method multiple times.
Creating a copy of a record	• Use the `copyEntity()` method to create a copy of a record.
Searching for a record	• Use the `search()` methods to search for a record: ○ One `search()` method uses the search criteria to search. ○ One `search()` method uses the search criteria and parameters to search. ○ One `search()` method uses an existing search view to search. • All `search()` methods use the search model classes.
Determining if security settings changed	• Use the `isSecurityChange()` method to determine if security settings for an entity changed.
Retrieves a record URL	• Use the `getRecordURL()` method to return the URL to a record on your instance.

For examples of some of these functions, see the code examples for each service class:

- AccountService
- AppointmentService
- ContactService
- DocumentService
- ExpenseService
- GroupService
- HistoryService
- InvoiceService
- ProjectService
- InvolvedService
- TaskService

- UserService

In the cases of the record that triggers the qualifier or action and the record with the custom block, you do not need to call the service class to access to the record.

The following table shows whether you use `ResourceService` classes in qualifiers, actions, and screens.

Services in Qualifiers, Actions, and Screens

Method	Qualifiers and Actions	Custom Screens
`read()` (for the initial record)	No	No
`create()`	No	Yes
`update()`	No	Yes
`new<object>()` (for the new object method in the Service class)	Yes	Yes

Qualifiers and Actions

When you create an automated qualifier or action, the code calls the `CustomCondition.condition(M)` or `CustomAction.action(M)` methods. Your code must override these methods, which allows access to the record that triggered the qualifier or action. Because you have access to the record, you do not need to call the service class to read the record.

Rules, wizards, and conditions save all changes automatically. As a result, you do not need to use the `create()` or `update()` service methods. To delete an object or create an object using the `new<Object>()` method in a service class, you must still call the service class for the object.

Custom Screens

When you create a custom block for a record, you use the `CustomBlock.getRecord()` method to access the record. Because you have access to the record, you do not need to call the service class for the record.

Unlike with automated qualifiers and actions, custom screens do not save automatically. As a result, if you want to create a new record or update the record with the screen, you must use the `create()` or `update()` methods. Because `create()` and `update()` are `ResourceService` methods, you must call the service class of the record as part of the code.

For example, if the custom screen is on the contact object definition, but you are not creating a new record or updating an existing contact, you do not need to call `ContactService`. However, if you want to update a field in the contact record when another field changes, your custom screen code must include the call to `ContactService` so that you can use the `update()` method.

When you want to delete a record that is a dependency for another record, you must first delete the record that has the dependency, then you can delete the dependency record. For example, if you want to delete a contact that is listed as the contact for a user, delete the user first. Then you can delete the contact. Because you must use the `delete()` method to perform the deletions, you must call `UserService` and `ContactService`.

However, if you want to delete a record that is a dependency for another record as part of an automated qualifier and action, you cannot delete both as part of the qualifier or action because both deletions cannot be part of the same transaction.

1.4.2.2.3.2 Other Service Classes

Any service classes that are not `ResourceService` classes initiate actions not specific to any model objects. These classes corresponds to different high-level functions in TeamConnect, including settings, security, and internationalization.

For examples of using these functions in the API, see the topics for the following classes:

- `CategoryService`
- `InternationalizationService`
- `LookupItemService`
- `ObjectDefinitionService`
- `ScheduledActionService`
- `SecurityService`
- `SettingsService`
- `UtilityService`

1.4.2.3 Getting Started with Custom Classes

When you create a rule in the **Setup** or use Screen Designer, TeamConnect provides customization options in the user interface. If the user interface does not offer the type of customization you want, you use the API classes to create custom code and upload the code to TeamConnect. The TeamConnect API allows you to create the following types of custom code:

- Conditions—You add conditions to the **Qualifier** page of a rule or the **General** tab of a condition. Custom conditions are also known as automated qualifiers.
- Actions—You add actions to the **Action** page of a rule and the **Actions** page of a wizard. Custom action are also known as automated actions.
- Blocks—You upload custom blocks to the **Screens** directory under the **Documents** tab and select them from the **Blocks** tab to add them to a specific object definition. Custom blocks are also known as custom screens.
- Wizard blocks—You upload custom blocks to the **Screens** directory under the **Documents** tab and select them from the wizard's **Page Components** tab. Wizard blocks are also known as wizard custom screens.
- Scheduled actions—You upload scheduled actions to the **Scheduled Actions** directory under the **Documents** tab and select them from the **Scheduled Actions** tool under the **All** tab.

- Tools—You upload custom tools to the **Tools** directory under the **Documents** tab and select them from the **Administer Custom Tools** page.

Use the sample code in this topic as a starting place for each previously mentioned code type.

Custom Action

This code provides a sample for a custom action on a contact rule. It also includes code for logging Debug messages.

```
import com.mitratech.teamconnect.enterprise.api.custom.CustomAction;
import com.mitratech.teamconnect.enterprise.api.model.Contact;

public class ContactCustomAction extends CustomAction<Contact> {

    @Override
    public void action(final Contact contact) {
        logDebug("contact="+contact);
    }
}
```

Custom Action with Parameters

This code provides a sample for a custom action with parameters on a contact rule.

```
import com.mitratech.teamconnect.enterprise.api.custom.CustomAction;
import com.mitratech.teamconnect.enterprise.api.model.Contact;

public class ContactCustomAction extends CustomAction<Contact> {

    @Override
    public void declareParameters() {
        getParameters().addTextParameter("name", "Label", "Default Value");
    }

    @Override
    public void action(final Contact contact) {
        logDebug("Parameter="+getParameters().getTextParameterValue("name"));
    }
}
```

Custom Condition

This code provides a sample for a custom condition on a contact rule or condition. It also includes code for logging Debug messages.

```
import com.mitratech.teamconnect.enterprise.api.custom.CustomCondition;
import com.mitratech.teamconnect.enterprise.api.model.Contact;

public class ContactCustomCondition extends CustomCondition<Contact> {

    @Override
    public boolean condition(final Contact contact) {
        logDebug("contact="+contact);

        return true;
    }
}
```

Custom Block

This code provides a sample for a custom block for a contact record. It also includes code for custom initialization.

```
import com.mitratech.teamconnect.enterprise.api.custom.CustomBlock;
import com.mitratech.teamconnect.enterprise.api.model.Contact;

public class MyCustomBlock extends CustomBlock<Contact>
{
    @Override
    public void initialize(java.util.Map<String, String> pageArgs) {
        // Perform some custom initialization
    }

    public String getText() {
        Contact contact = getRecord();
        // Display something on screen

        return contact.getDisplayString();
    }
}
```

Wizard Custom Block

This code provides a sample for a custom block for a contact wizard. It also includes code for parameters and custom initialization.

```
import com.mitratech.teamconnect.enterprise.api.custom.WizardCustomBlock;
import com.mitratech.teamconnect.enterprise.api.model.Contact;
```

```
public class MyCustomBlock extends WizardCustomBlock<Contact> {
    @Override
    public void initialize(java.util.Map<String, String> pageArgs) {
        // Perform some custom initialization
        getParameters().addTextParameter("name", "Label", "Default Value");
    }

    public void action() {
        Contact contact = getRecord();
        logDebug("Parameter=" + getParameters().getTextParameterValue("name"));
        logDebug("Contact=" + contact.getDisplayString());
    }
}
```

Scheduled Custom Action

This code provides a sample for a scheduled custom action for a contact. It also includes code for logging Debug messages.

```
import com.mitratech.teamconnect.enterprise.api.model.Contact;
import com.mitratech.teamconnect.enterprise.api.custom.ScheduledCustomAction;

public class SampleActionClass extends ScheduledCustomAction {
    @Override
    public void action() {
        logDebug("message");
    }
}
```

Custom Tool

This code provides a sample for a custom tool. It also includes code for logging Debug messages.

```
import com.mitratech.teamconnect.enterprise.api.custom.CustomTool;
import com.mitratech.teamconnect.enterprise.api.model.Contact;

public class MyCustomTool extends CustomTool {
    @Override
    public void initialize(java.util.Map<String, String> pageArgs) {
        // Perform some custom initialization
    }
```

```
    public void action() {
        Contact contact = getRecord();
        logDebug("Contact=" + contact.getDisplayString());
    }
}
```

1.4.2.3.1 Automated Qualifiers and Actions

Automated qualifiers are files that you create to add conditional expressions to TeamConnect rules and conditions. Automated actions are files that you create to add actions to rules and wizards.

Automated qualifiers and actions display in one of the following two states:

- **Current Object State—**Object data displayed on the screen at the time when the user attempts a specified operation. In existing records, this data may include previously saved values from the database, as well as the values entered by the user before updating the record.

  In new records that are being created, this data includes only the values entered by the user and automatically populated by the system. All rule types, except security, typically use the current object data. Post commit rules only use the current object state.

- **Old Object State—**Object data stored in the database at the time when the user attempts a specified operation. This applies only to existing records and excludes the values that the user may have entered before saving the record.

  Security rules are the only rule type that uses exclusively the old object state. However, sometimes, other rule types may use the old object state to check if any changes have been made to the current object.

Rule Types and Execution Order

The following table indicates which rules may have custom or automated qualifiers and actions.

Rule Types

Rule Type	Can Have Automated:		Execution Order
	Qualifier	Action	
Security	x		1
Pre-population	x	x	2
Validation	x		3
Approval	x		4
Custom Action	x	x	5

Scheduled Action	**x**	**x**	**6**
Audit	**x**	**x**	**7**
Post Commit	**x**	**x**	**8**

When a user attempts an operation in TeamConnect, multiple rules may trigger. If rules of different types apply to that operation, the rules execute according to the predefined order indicated in the Rule Types table. You cannot modify the predefined order in which multiple rules execute.

If several rules of the same type trigger, they execute according to the order number of the rule in the user interface (for example, see the General Tab on Rule Screens image). You can change this order.

Parameters

Parameters are arguments passed to the rule through fields in the user interface. The Java file defines how parameters appear as fields in the user interface. The user specifies parameter values in the rule interface rather than in the file. As a result, you can maintain the rule components without modifying the actual code.

1.4.2.3.1.1 Automated Qualifiers

You add qualifiers to the **Qualifier** page of a rule and the **General** page of a condition. All qualifiers return a Boolean to determine whether the qualifier conditions are true or false.

When qualifiers are automated, they use Java files to create the qualifier. Automated qualifiers for rules define whether the associated action executes; the action executes if the qualifier is true. Automated qualifiers for conditions define whether the condition is true.

After you create an automated qualifier, upload the file to the **Automated Qualifiers** folder in the appropriate object definition. For more details, see Uploading Rule Component Files.

When to Create Automated Qualifiers

You create an automated qualifier when you cannot create the qualifier through the user interface. Create an automated qualifier in the following situations:

- You need access to more methods than those methods available through the TeamConnect object model.

  For example, a rule may require that an action executes if a project has a related involved party record with a specific role. Because projects have no one-to-many relationship with involved parties, creating this rule requires an API method to return the list of the associated involved party records.

  For a sample in Java, see Checking Category List in Related Records.

- You need to use Boolean operators, such as NOT, XOR, NOR, XNOR.

For more automated qualifier samples, see Automated Qualifiers.

Wizard Page Transition Rules

Page transition rules are similar to the TeamConnect rule types in the Rule Types table, especially validation rules. They determine the sequence of the wizard pages by performing actions based on conditions met in the qualifier. Along with values of the main object, automated qualifiers in page transition rules may use wizard parameters and values from related objects and sub-objects defined in the templates associated with the wizard.

You cannot customize actions in page transition rules because they have a predefined transition to a specific page in the wizard. All page transition rules also have the same predefined trigger, which you do not need to define. The trigger is the transition between pages that occurs when you click the **next** button on the wizard page for the rule.

For example, if the wizard identifies a claimant, a page transition rule may use the provided information to search the database for the contact. If the system finds a matching contact, the wizard goes to the page that verifies the available contact information. If not, the wizard goes to the page that collects the contact information for a new contact record.

1.4.2.3.1.2 Automated Actions

You add actions to the **Action** page of a rule and the **Actions** page of a wizard. Rule actions specify what occurs if the rules meet the conditions in rule qualifiers. Wizard actions specify the actions for each wizard page. Unlike automated qualifiers that must return a boolean value, actions do NOT return values.

Automated actions files for both rules and wizards must be uploaded to the **Automated Actions** folder in the appropriate object definition. For more details, see Uploading Rule Component Files.

Rule Automated Actions

Rule actions execute when qualifiers return true. With automated rule actions, you can create new records and compare, calculate, and insert values. You use automated actions for custom and scheduled action rules.

For example, a rule may require that if a contact name changes, a history record for the contact saves with the date when the name changed and the ID of the user who made the change. For rule action samples in Java, see Rule Actions.

You do not create automated actions in the following situations:

- To deny the operation that triggers a rule.
- To approve the operation that triggers a rule and its route.
- To define actions with a template, such as populating fields with defined values.

Wizard Automated Actions

Wizard actions execute when the user clicks **next** on the wizard page with the action, moving the wizard to the next page. Like rule actions, automated wizard actions create new records and compare, calculate, and insert values.

For example, in a wizard, you may want to assign a value to a custom field of the object. Or, instead of assigning values you may want to calculate values, display additional entry fields, overwrite values

in the associated templates, validate the entered data, or create a new record. For examples of wizard page actions in Java, see Wizard Page Actions.

When writing automated page actions, make sure they will not duplicate custom action rules with the **Create** trigger. Custom action rules execute when the wizard finishes and the rule qualifiers are met.

You do not need to create automated actions in wizards when you need to assign a value to a system field of the main object, its sub-objects, and related objects.

Important: *In wizards, do not confuse page actions with rule actions in page transition rules because you cannot automate actions in page transition rules.*

1.4.2.3.1.3 Qualifier and Action Code

When creating automated qualifiers and automated actions, you use the following subclasses of `CustomItem`:

- `CustomCondition`—Use when creating automated qualifiers for rules and conditions.
- `CustomAction`—Use when creating automated actions for rules and wizards.

The following table provides the basic differences between qualifiers and actions.

Qualifiers and Actions

Qualifiers	Actions
Extend `CustomCondition` and the class of the enterprise entity,	Extend `CustomAction` and the class of the enterprise entity,
Implement the abstract method `CustomCondition.condition(final M record)` that returns true or false.	Implement the abstract method `CustomAction.action(final)` that does not return anything.
Define your logic in the `condition(M)` method. Because the qualifier returns a Boolean value, if the return value is true for rules, TeamConnect runs the action associated with this condition. If it is false, the action does not execute.	Define your logic in the `action(M)` method.

You call the `condition(M)` and `action(M)` methods when the qualifiers and actions run. Override these methods to access an instance of the object.

The following code snippet demonstrates the basic structure of a qualifier in Java. You must determine what else to import for your qualifier.

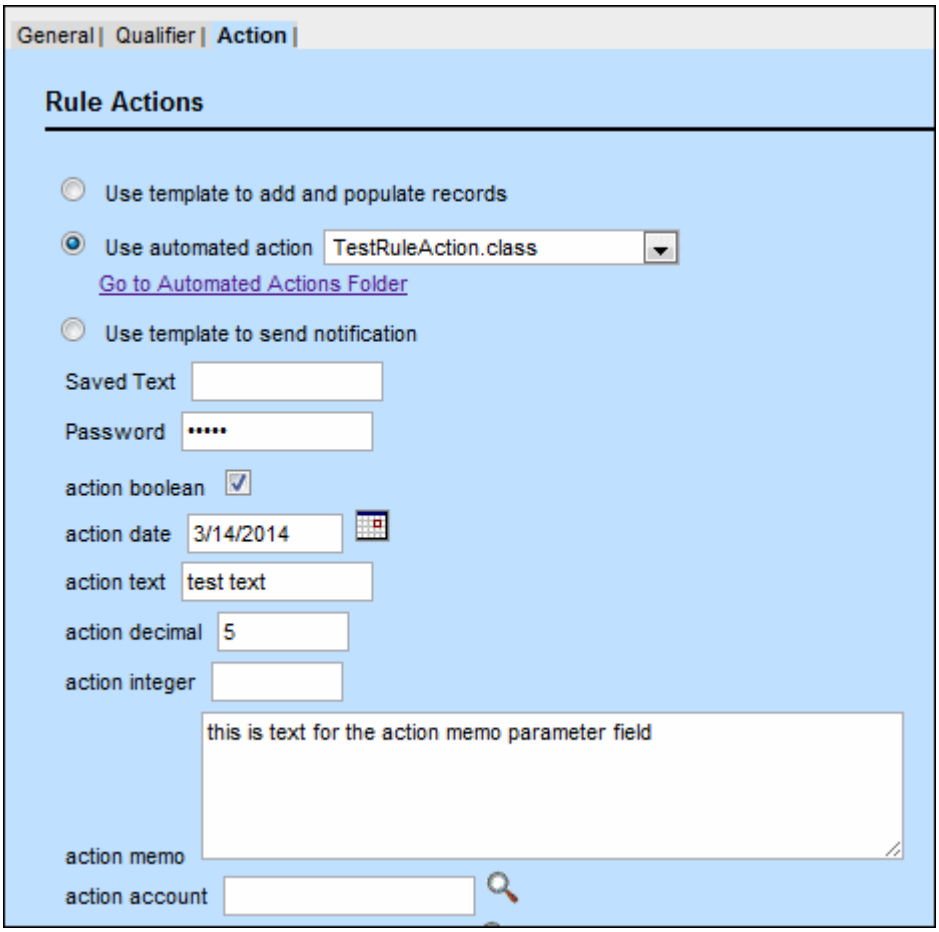
```
import com.mitratech.teamconnect.enterprise.api.model.Project;
import com.mitratech.teamconnect.enterprise.api.model.rule.CustomCondition;
public class MyCustomCondition extends CustomCondition<Project>
```

```
{
    @Override
    public boolean condition (final Project record){
        // Put your code here
    }
}
```

The following code snippet demonstrates the basic structure of an action in Java. You must determine what else to import for your action.

```
import com.mitratech.teamconnect.enterprise.api.model.Project;
import com.mitratech.teamconnect.enterprise.api.model.rule.CustomAction;
public class MyCustomAction extends CustomAction<Project>
{
    @Override
    public boolean action (final Project record) {
        // Put your code here
    }
}
```

Parameters in actions and qualifiers are values that the Java file displays as fields in the user interface, as shown in the following image.



Parameters Added from an Automated Action

If values in a qualifier or an action might change, use parameters so that the solution developer can change the value without modifying the action or qualifier file. In addition, when you use parameters in rules, you are sometimes able to use the same file for multiple rules that require different values.

For example, when an automated action creates a task with the due date based on a date in a project, the solution developer can set the number of days before the task is due through the user interface. You can also use an automated action to create new accounts for budgets that a solution developer sets from a parameter.

Set or get parameter values using the methods from the `Parameters` class.

For code samples, see Parameterized Actions.

About Parameters in Wizards

Wizard page actions use the following types of parameters:

- Wizard parameters that are transient values of different types that the wizard creates, uses, and discards afterwards. The user who designs the wizard through the user interface defines the wizard parameter name within the wizard.

- All sub-objects and related objects defined in the template, which the wizard treats as parameters. The system within the template automatically defines the names of these parameters, using the name of the object and a sequential number. For example, Account 1, History 1, Assignee 1, and Assignee 2.
- Parameters that you use in any automated action or qualifier.

Adding and Using Parameters

Qualifiers and actions both accept parameters. Parameters are fields that appear on the **Qualifier**, **General**, and **Action** pages that someone working in the **Setup** can update. When adding, getting, or setting parameters, use the `Parameters` class.

You define parameters in rules in the following way:

- The `CustomCondition.declareParameters()` or `CustomAction.declareParameters()` methods, which accept methods from the `Parameters` class.
- The add methods in the `Parameters` class, which create the parameters. The class includes a parameter for each data type and certain system objects. For example, if you want to add a project field to the **Qualifier** or **Action** page, use the `addProjectParameter()` method.

After you define parameters, you include the code for the qualifier or action using the `condition(M)` or `action(M)` methods. As part of this code, you can retrieve the value entered for a parameter in the **Setup**. In the case of the project parameter example, you use the `getProjectParameterValue()` method to retrieve the project. If you need to update the value an existing parameter within the `condition(M)` or `action(M)` methods, you can use the set methods. For example, `setProjectParameterValue()`.

If no one enters a value for the parameter, the qualifier or action uses the default value for the parameter. When someone enters a value in the **Setup** for the parameter field, the entered value overrides the default value.

The following code sample demonstrates how to add parameters to the **Action** page of a rule. The action code retrieves the value of the parameter text field with the name "text" and updates the name of a project with the value.

```
public class TestRuleAction extends CustomAction<Project> {


    @Override
    public void declareParameters() {
        parameters.addTextParameter("text", "Text", "default value");
        parameters.addPasswordParameter("password", "Password", "default
        password");
        parameters.addBooleanParameter("action boolean", "action boolean",
        false);
        Date date = new Date();
        parameters.addDateParameter("new action date", "new action date", new
        Date());
        parameters.addDecimalParameter("action decimal", "action decimal",
        BigDecimal.valueOf(.5));
        parameters.addNumberParameter("action number", "action number",
        Long.valueOf(5));
```

```
        parameters.addMemoParameter("action memo", "action memo", "This is memo
        text field content");
        Account account = platform.getAccountService().read(Long.valueOf(503));
        addAccountParameter("action account", "action account", account);
    }

    @Override
    public void action(final Project dispute) {
        dispute.setName(parameters.getTextParameterValue("text"));
        Account newAccount = platform.getAccountService().newAccount("Test
        Account", BigDecimal.valueOf(1000), new Date(), new Date());
        newAccount.setNote("Here is a note that has been set");
    }
}
```

The sample code in this topic is for a qualifier that checks whether the policy holder of a Policy changed. An Involved custom field in the Policy custom object definition specifies the policy holder. If the primary key of the policy holder is the same as the primary key of the policy holder in the previous state of the record, the qualifier returns true, and the action runs. Otherwise, the qualifier returns false.

Before attempting to run this rule, you must create a custom field of type Involved in the root of the Policy custom object definition. Then, you specify this custom field as a parameter for the qualifier.

Defining the Parameter

As explained in Parameters in Qualifiers and Actions, all rule parameters use the `Parameters` class and the `declareParameter()` method of a rule condition or action. Because the example rule compares the contents of an involved custom field in two states of a record, we need to be able to locate that custom field. The `Parameter.addTextParameter()` method in the example locates the field by taking three arguments:

- The label of the parameter. This text describes the custom field.
- The name of the parameter. This text identifies the custom field.
- The default value of the parameter.

Specifying this parameter results in a text box with the appropriate label in the **Qualifiers** tab of the rule. The following code sample shows how to define this text box using a parameter.

```
import com.mitratech.teamconnect.enterprise.api.custom.CustomCondition;
public class MyParameterizedCustomCondition extends CustomCondition<Project>
{
    @Override
    public void declareParameters() {
        parameters.addTextParameter("My Parameter Label", "myParameter",
        "defaultValue");
```

```
    }
    @Override
    public boolean condition(final Project dispute) {
    // Place code for all conditions here.
}
```

Checking the Condition

After defining the parameter, the rule must be able to compare the policy holder in the previous version of the policy with the policy holder in the new version. The rule should return true if the primary keys of the custom fields' contents are the same. When writing the code, the rule must perform the following actions:

- Retrieve the associated project.
- Retrieve the old state of the record from the database.
- Retrieve the parameter using the name of the parameter, which is `fieldKey` for this sample.

  Use the `getTextParameterValue()` method to retrieve the value: `String fieldKey = parameters.getTextParameterValue("fieldKey");`

Completing the Qualifier

The following code sample includes the parameter and conditions:

Completed Qualifier Example

```
import com.mitratech.teamconnect.enterprise.api.custom.CustomCondition;
import com.mitratech.teamconnect.enterprise.api.model.Contact;
import com.mitratech.teamconnect.enterprise.api.model.Project;
public class CheckingWhetherPolicyHolderWasChanged extends
CustomCondition<Project>
{
    @Override
    public void declareParameters() {
        parameters.addTextParameter("Policy custom field name",
        "policyCustomFieldName", "policyHolder");
    }
    @Override
    public boolean condition(final Project dispute) {
        // Getting the old object.
        Project oldDispute = platform.getProjectService().read(dispute);
        // Getting parameter value that indicates custom field key
        String policyCustomFieldName =
        parameters.getTextParameterValue("policyCustomFieldName");
        // Getting policy holder from the old object's custom field.
        Contact oldPolicyHolder =
        oldDispute.getInvolvedFieldValue(policyCustomFieldName).getContact();
```

```
        // Getting policy holder from the current object's custom field.
        Contact currentPolicyHolder =
        dispute.getInvolvedFieldValue(policyCustomFieldName).getContact();
        // If the policy holder in the old object is not the policy holder in the
        current object, return true.
        return
        !oldPolicyHolder.getPrimaryKey().equals(currentPolicyHolder.getPrimaryKey
        ());
    }
}
```

1.4.2.3.2 Custom Pages

Use XML files to create custom screens and tools. The XML file includes the structure, layout, and contents of a custom block. If the screen or tool has complex business requirements or if data is pulled from related records, you can use a Custom Java Block (CJB). The CJB uses the TeamConnect API to specify actions and properties for the custom block.

Note: *You do not need a CJB if the page is only referencing system fields or existing custom fields or you just want to display plain HTML.*

When implementing a CJB, stay within the following JavaBean conventions:

- The get method signature—`public PropertyType getPropertyName()` or `public boolean isPropertyName()`

  For example:

  ```
  public String getName();

  public boolean isActive();
  ```

- The set method signature—`public void setPropertyName(PropertyType value)`

  For example:

  ```
  public void setName(String name);

  public void setActive(boolean isActive);
  ```

Service and Model Classes

When you add a custom screen or tool to an object, you create page sections. When custom page requires information, access the associated service classes and call specific model methods in the CJB.

For custom screens only, because the underlying record is already available to the CJB because of the `getRecord()` method, you do not need to use the `read()` method in the record service class to access the record. If you need to access a different record of the same type, then you access the service class. For example, if you create a contact record as part of the screen, access `ContactService` in the code to use the `create()` method.

Note: *If you want to perform some initialization when TeamConnect first loads a screen or tool, you must override the initialize() method. For example, if you want to access the page arguments that launch a wizard, you can use the initialize() method to access them.*

OnClick and OnChange Functions

If the custom block includes a CJB, you execute actions in the CJB from the XML file using the following functions:

- invokeCustomAction—Use in custom screen XML files.
- invokeToolAction—Use in custom tool XML files.

Use these functions for onClick and onChange events. For example, you can use the `invokeCustomAction` function to execute an action when a user clicks a button in a screen, as shown in the following sample:

```
<input type="button" value="Execute action1" onClick="invokeCustomAction(this,
'cjb', 'action1', 'hi', 1, true)" />
```

In the previous sample, invokeCustomAction takes the following parameters:

- `this`—The first parameter of these functions in all custom XML files.
- `'cjb'`—The ID of the CJB defined in the XML file.
- `'action1'`—The name of the Java method in the CJB.
- `'hi', 1, true`—Arguments that the action uses.

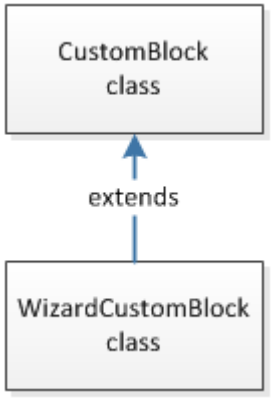
If this button was in a custom tool, the parameters of the `invokeToolAction` function would include the following two parameters:

- `'action1'`—The name of the Java method in the CJB.
- `'hi', 1, true`—Arguments that the action uses.

1.4.2.3.2.1 Custom Screens

Custom screens extend the `CustomBlock` class and are made up of custom blocks. Any block that is not part of the original functionality of TeamConnect is a custom block. Wizard custom blocks are a type of custom block that you create for wizards.

The following diagram illustrates how custom screens work in the API.



Refer to the code sample for an example of a custom screen.

Wizard Custom Screens

As shown in the previous diagram, the `WizardCustomBlock` class extends the `CustomBlock` class when you are creating a screen for a wizard. As a result, any custom block can be a wizard custom block. However, you cannot use a wizard custom block for a custom screen because wizard blocks may contain wizard parameters.

You can use wizard parameters in two ways:

- *To pass data from one page to another.*

  When you add a wizard parameter to pass data, the parameter does not automatically appear in the wizard block.

  To add a wizard parameter, use the `Parameters` object method for adding the specific type of parameter. For example, the following code adds text and boolean parameters.

```
getParameters().addTextParameter("text", "Text", "default value");
getParameters().addBooleanParameter("action boolean", "action boolean",
false);
```

  After you add the parameter to the code, you can refer to it on any other wizard page by the String name of the parameter. Retrieve the parameter using the get method for the parameter type. In the case of the previous two parameters, use the following code to retrieve the parameters.

```
getParameters().getTextParameterValue("text");
getParameters().getBooleanParameterValue("action boolean");
```

- *To display a parameter field on a wizard custom screen.*

  To add a parameter to the wizard screen, you must define it on the **Page Components** page in the **Setup**. From the XML file, you can use the tc:WizardParameter tag to display the parameter on the wizard custom screen.

The code samples in this section provide an example of how to use CJB files to display the following block in a contact object. When you click one of the buttons in the screen, the system creates or updates a history entry.

Custom Block Example

CJB File

The following code sample provides the CJB file for the custom block example.

```
import com.mitratech.teamconnect.enterprise.api.custom.CustomBlock;
import com.mitratech.teamconnect.enterprise.api.model.Company;
import com.mitratech.teamconnect.enterprise.api.model.Contact;
import com.mitratech.teamconnect.enterprise.api.model.History;
import com.mitratech.teamconnect.enterprise.api.model.Person;
import com.mitratech.teamconnect.enterprise.api.service.HistoryService;

// Contact CJB example
public class MyCustomBlock extends CustomBlock<Contact>
{
    @Override
    public void initialize(java.util.Map<String, String> pageArgs) {
        // Perform some custom initialization
    }

    // Display something on the screen (see XML)
    public String getText() {
```

```
        Contact contact = getRecord();
        if (contact instanceof Person) {
            return "Hi " + ((Person) contact).getFirstName();
        } else {
            return "Hi " + ((Company) contact).getName();
        }
    }

    // Execute a user action (see XML)
    public void action0() {
        if (isDebug()) {
            logDebug("action0 executed");
        }
        // Create a History
        HistoryService historyService = platform.getHistoryService();
        History newHistory = historyService.newHistory("Action0 button was
        clicked by " +
        platform.getUtilityService().getCurrentUser().getDisplayName(),
        getRecord());
        historyService.create(newHistory);
    }

    // Execute a user action with arguments (see XML)
    public void action1(String arg1, int arg2, boolean arg3) {
        try {
            // Update History
            History lastHistory =
            platform.getHistoryService().getLastHistory(getRecord());
            lastHistory.setNote(String.format("Arguments passed to method were
            arg1=%s, arg2=%d, arg3=%b", arg1, arg2, arg3));
            platform.getHistoryService().update(lastHistory);
        } catch (Exception e) {
            logError("Error in action1", e);
        }
    }
}
```

Here's how the previous CJB sample works:

- The beginning of the code extends `CustomBlock`. When you extend `CustomBlock`, the object type must match the object for which you are defining the CJB file. Because the code in this sample is for the contact object, the CJB file extends `CustomBlock<Contact>` with the following code.

```
public class MyCustomBlock extends CustomBlock<Contact>
```

- The body of the `initialize()` method includes any initial processing. The sample overrides the `initialize()` method to access the page arguments that launch the screen.

```
@Override
public void initialize(java.util.Map<String, String> pageArgs)
```

- The sample retrieves the contact record with the custom block.

```
Contact contact = getRecord();
```

Note: *Because the CJB retrieves the contact record with this code, it does not need to performing a ResourceService read() action.*

- The following code displays the contact name fields on the screen after retrieving the object.

```
public String getText() {
    Contact contact = getRecord();
    if (contact instanceof Person) {
        return "Hi " + ((Person) contact).getFirstName();
    } else {
        return "Hi " + ((Company) contact).getName();
    }
}
```

- When the user clicks the action0 button, the following code executes in response to the user action:
  - The system checks if the logger for the custom code is enabled for the **Debug** level. If true, the system logs a message.

```
if (isDebug()) {
    logDebug("action0 executed");
}
```

  - The code accesses the HistoryService class using the `Platform` class.

```
HistoryService historyService = platform.getHistoryService();
```

  - The system creates a new history entry with the name of the user who clicked the button.

```
History newHistory = historyService.newHistory("Action0 button was
clicked by " +
platform.getUtilityService().getCurrentUser().getDisplayName(),
getRecord());
historyService.create(newHistory);
```

- When the user clicks the action1 button, the following code executes with arguments in response to the user action:
  - The action lists the three arguments.

```
public void action1(String arg1, int arg2, boolean arg3) {
```

  - The system tries retrieves the most recent history and updates the note of the entry.

```
try {
    // Update History
    History lastHistory =
    platform.getHistoryService().getLastHistory(getRecord());
    lastHistory.setNote(String.format("Arguments passed to method were
    arg1=%s, arg2=%d, arg3=%b", arg1, arg2, arg3));
    platform.getHistoryService().update(lastHistory);
}
```

  - If the system cannot perform an update, the code logs an error.

```
catch (Exception e) {
    logError("Error in action1", e);
}
```

XML File

The following code sample provides the XML file for the custom block example. See Creating Block XML Files for more information about creating XML files.

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:transform version="4.0" xmlns:tc="http://www/w2.prg/1999/XSL/Transform">
   <tc:blockTemplate>
       <tc:useClass id="cjb" name="MyCustomBlock"/>
           <!-- Display a system field -->
           ID: <tc:field  name="enterpriseEntity.idNumber" />
           <br/><br/>
           <!-- Display a dynamic value -->
           CJB text: <tc:out value="${cjb.text}" />
           <br/><br/>
           <!-- Execute a cjb action without arguments -->
           <input type="button" value="Execute action0"
           onClick="invokeCustomAction(this, 'cjb', 'action0')" />
           <br/><br/>
           <!-- Execute a cjb action with arguments -->
           <input type="button" value="Execute action1"
           onClick="invokeCustomAction(this, 'cjb', 'action1', 'hi', 1, true)" />
```

```
    </tc:blockTemplate>
</tc:transform>
```

Here's how the previous XML file works:

- The tc:useClass tag references the Java class, as shown in the following code.

  ```
  <tc:useClass id="cjb" name="MyCustomBlock"/>
  ```

  This tag includes the following attributes:

  - `name`—The name of the Java class.
  - `id`—An alias for the Java class. The invokeCustomAction function refers to the Java class using this alias.

- The following code displays a number system field and a text field with a dynamic value.

  ```
  ID: <tc:field  name="enterpriseEntity.idNumber" />
  CJB text: <tc:out value="${cjb.text}" />
  ```

- The following code executes an action without arguments and an action with arguments. This code provides an example of how to use the invokeCustomAction function. When you click a button, you call invokeCustomAction, which submits the page and causes the method in the Java class to execute. Methods can also execute with arguments, as shown in the second invokeCustomAction function.

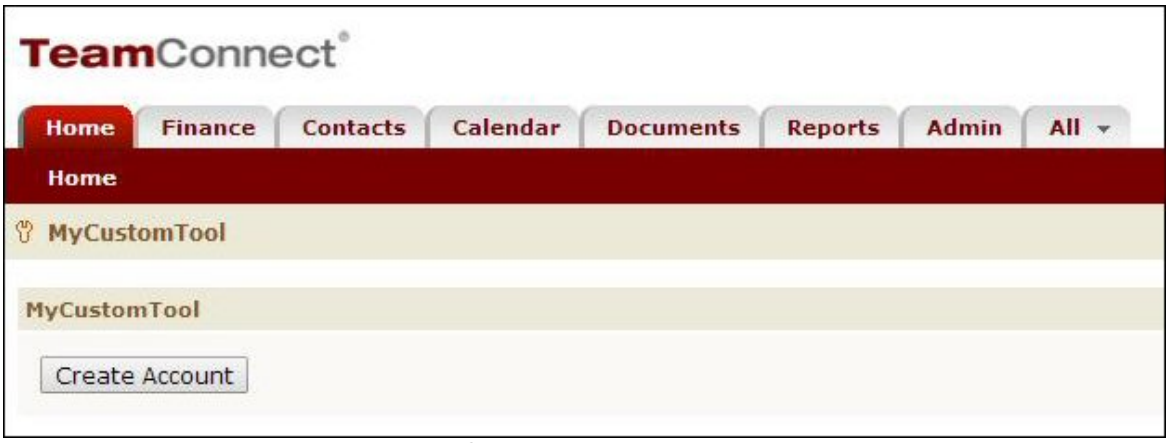
  ```
  <input type="button" value="Execute action0"
  onClick="invokeCustomAction(this, 'cjb', 'action0')" />
  <input type="button" value="Execute action1"
  onClick="invokeCustomAction(this, 'cjb', 'action1', 'hi', 1, true)" />
  ```

1.4.2.3.2.2 Custom Tools

Custom tools extend the `CustomTool` class. Code for custom tools is similar to code for custom screens.

Refer to the code sample for an example of a custom tool.

The code samples in this section provide an example of how to use CJB files to display the following block in a custom tool.



Custom Tool Example

Java File

The following code sample provides the CJB file for the custom block example.

```
import com.mitratech.teamconnect.enterprise.api.custom.CustomTool;
import com.mitratech.teamconnect.enterprise.api.model.Account;

public class MyCustomTool extends CustomTool
{
    public void createAccount() {
        String accountName = "New Account";
        BigDecimal allocationLimit = new BigDecimal(1000);
        Date startDate = new Date();
        Date endDate = DateUtils.addYears(startDate, 1);
        Account newAccount = platform.getAccountService().newAccount(accountName,
        allocationLimit, startDate, endDate);
        platform.getAccountService().create(newAccount);
    }
}
```

Here's how the previous CJB sample works:

- The beginning of the code extends `CustomTool`, as shown in the following code.

```
public class MyCustomTool extends CustomTool
```

- The sample accesses the AccountService class using the `Platform` class and creates a new account object called **New Account**.

```
String accountName = "New Account";
```

```
BigDecimal allocationLimit = new BigDecimal(1000);
Date startDate = new Date();
Date endDate = DateUtils.addYears(startDate, 1);
Account newAccount = platform.getAccountService().newAccount(accountName,
allocationLimit, startDate, endDate);
```

- The sample executes the `create()` method to create a new account from the tool.

```
platform.getAccountService().create(newAccount);
```

XML File

The following code sample provides the XML file for the custom tool example. See Creating Block XML Files for more information about creating XML files.

```
<?xml version="1.0" encoding="UTF-8"?>
<tc:transform version="1.0" xmlns:tc="http://www.w3.org/1999/XSL/Transform">
    <input id="createAccount" name="createAccount" type="button"
    onclick="invokeToolAction('createAccount', 'createAccount');"
    title="<teamconnect:message key='button.yes'/>" value="<teamconnect:message
    key='button.yes'/>" />
</tc:transform>
```

Here's how the previous XML sample works:

- When you save this XML file, you give it the same name as the Java file. The system knows that the XML file refers to the Java file of the same name.
- The <input> tag includes the following parts:
  - The ID, the name, and the type of `<input>` tag.

    ```
    id="createAccount" name="createAccount" type="button"
    ```

  - The onClick event with the invokeToolAction function. When you click the button, you call `invokeToolAction`, which submits the page and causes the method in the Java class to execute.

    ```
    onclick="invokeToolAction('createAccount', 'createAccount');
    ```

  - The title and value of the button with a custom key that you can internationalize. To use this code, replace *button.yes* with a custom internationalization key for this button.

    ```
    title="<teamconnect:message key='button.yes'/>"
    value="<teamconnect:message key='button.yes'/>"
    ```

1.4.2.3.3 Scheduled Actions

The **Scheduled Actions** tool controls actions that are scheduled to execute at specific times in the future. When you add a new scheduled action from the tool, you can select a Java class from the **Action** drop-down.

All scheduled actions that you create through the API automatically save to the **Scheduled Actions** tool.

In the API, the following interfaces represent scheduled actions:

- `ScheduledActionService`—Includes methods that you use to create, delete, activate, and retrieve scheduled actions.
- `ScheduledActionStatus`—Includes methods that you use to return information about each scheduled action.
- `Parameter`—Includes methods that you use to return information about parameter fields for the scheduled action.
- `Schedule`—Includes methods that you use to return information about the start date, end date, and how often the scheduled action occurs.

1.4.2.3.3.1 Adding Java Classes for Scheduled Actions

You use the API to create a Java class for the scheduled action. After you have a file for the Java class, you can add it to the **Scheduled Actions** folder in TeamConnect and select it from the **Actions** drop-down list.

The action in the following code sample retrieves a text parameter, creates a new person contact, and adds an address to the contact.

```
import com.mitratech.teamconnect.enterprise.api.model.Contact;
import com.mitratech.teamconnect.enterprise.api.model.Address;
import com.mitratech.teamconnect.enterprise.api.service.ContactService;
import com.mitratech.teamconnect.enterprise.api.custom.ScheduledCustomAction;

public class SampleActionClass extends ScheduledCustomAction {

    @Override
    public void action() {
        //Retrieve the contact service from platform.
        final ContactService contactService = platform.getContactService();

        //retrieves a text parameter
        String paramCity = getParameters().getTextParameterValue("city");

        //create a new contact of type person.
        Contact cont = contactService.newPerson("firstName", "lastName");
        //add a business address to the new contact.
```

```
        Address address =
        cont.addAddress("BUS1").setCity(paramCity).setState("TX");


        //create new contact
        Contact createdContact = contactService.create(cont);
    }
}
```

1.4.2.3.3.2 Using the API with Scheduled Actions

You can use `ScheduledActionService` to do most of the actions you can do on the **Scheduled Actions** tool page. Also like the tool, you cannot update scheduled actions. You can only delete or deactivate existing actions.

When you create a scheduled action using the API, you reference a Java class in the **Scheduled Actions** folder. If the Java class is not in the folder, you can add it through the API using `DocumentService`. The name of the Java class in the following code sample is `ActionClass`.

```
public class MyRule extends CustomAction<Project> {


    @Override
    public void action(Project project) {
        //the action class for the scheduled action.
        Class<ActionClass> actionClass = ActionClass.class;


        // Retrieve the service
        ScheduledActionService scheduledActionService =
        platform.getScheduledActionService();


        // Build parameters
        Parameters parameters = scheduledActionService.newParameters();
        parameters.addBooleanParameter("isAThing", "This is a thing", false);
        parameters.addProjectParameter("triggeringProject", "Project responsible
        for the action", project);


        //create a new weekly schedule.
        Calendar cal = Calendar.getInstance();
        cal.set(2014, 06, 19);
        Schedule schedule = Schedule.createWeekly(cal.getTime(), 2);
        ScheduledActionStatus generatedUniqueId =
        scheduledActionService.scheduleAction(actionClass, parameters, schedule);
    }
}


public class ActionClass extends ScheduledCustomAction {
```

```
    @Override
    public void action() {
        // Repeated action code
    }
}
```

If you want to get a scheduled action from the tool, you can use the `getScheduledAction()` method, as shown in the following sample.

```
ScheduledActionStatus scheduledActionStatus =
scheduledActionService.getScheduledAction("uniqueName");
```

If you want to deactivate a scheduled action using the API, you can use the `deactivate()` method, as shown in the following sample.

```
scheduledActionService.deactivate("uniqueName");
```

1.4.2.4 Creating New Records

Specific conditions frequently require the creation of new records. For example, a business rule may require that when a value changes in a project, the rule automated action to create a new history record.

When creating new records, you use the following methods:

- The `new<Object>()` methods in the resource service classes to create new objects. For example, `newExpense()` in `ExpenseService`.

  Note: *When writing code to create a new record for a rule, you do not need to use the create() method because the rule automatically commits the changes to the database.*

- The `ResourceService.create()` method to save the created object to the database after you create the empty object.
- Object-specific methods that you can set in the new empty object using the API model interfaces for the objects.
- The `UtilityService.runAsSystemUser()` method to create the record regardless of the current user's rights. For more details, see Running Code as the System User.

In addition to the resource service samples, refer to Creating a History Record, Creating a Child Project, and Creating Related Task Record for more samples about creating a record in Java.

1.4.2.5 Object Definitions

An implementation of TeamConnect contains a variety of object definitions that are classified in one of the following ways:

- System Object Definitions—Objects that exist as part of the core functionality of TeamConnect, before any customization. A TeamConnect implementation may utilize some, all, or none of the system objects. System objects include Account, Appointment, Contact, Document, Expense, History, Invoice, Involved, and Task.
- Custom Object Definitions—Objects defined for a TeamConnect implementation. Custom objects could be Claims, Cases, Matters, Litigations, among others. They can have parent-child relationships with one another or you can define a custom object as an embedded object of another custom object, which is a child object with limited functionality.

An instance of an object is a record. For more details about system or custom objects, see About Objects.

ObjectDefinitionService Interface

If you want to retrieve object definitions using the API, you can use `ObjectDefinitionService`. As part of the `ObjectDefinitionService` interface, you can use the `getObjectDefinition()` method to retrieve an object definition. If the API cannot find an object definition that fits the unique code, the method throws an exception. The following code sample uses the `getObjectDefinition()` method to retrieve the contact object definition.

```
ObjectDefinition contactObjectDefinition =
objectDefinitionService.getObjectDefinition("CONT");
```

In the following code sample, you can use the `getObjectDefinition()` method to retrieve a custom object definition with the unique code of MATT.

```
ObjectDefinition matterObjectDefinition =
objectDefinitionService.getObjectDefinition("MATT");
```

Note: *Use the following code to access ObjectDefinitionService:* `ObjectDefinitionService objectDefinitionService = platform.getObjectDefinitionService();`

ObjectDefinition Interface

The `ObjectDefinitionService` methods use the `ObjectDefinition` class to return information about object definitions. `ProjectObjectDefinition` and `InvolvedObjectDefinition` also extends `ObjectDefinition`. The `ObjectDefinition` class includes the settings on the following pages of an object definition in the **Setup**:

- **General**—The `hasInvolved()`, `isContactCentric()`, and `isNewBtnNotShownInRelObjBlock()` methods refer to settings that might appear on the **General** page, depending on the type of object definition. If an object definition does not have the associated setting, the method returns `false`.
- **Wizards**—The `getRequiresWizardForCreate()` method refers to whether or not the setting on the **Wizards** page specifies that a wizard is required to create a new record.

1.4.2.5.1 System Objects

System objects are the out-of-the-box objects included with TeamConnect. System objects allow you to do basic functions, such as creating contacts or invoices.

All system objects in the API each have a resource service class. These resource service classes use the API model interfaces to work with system fields. All system fields have get and set methods in their owner interfaces. You can also get or set custom fields on system objects.

1.4.2.5.1.1 Accounts

Accounts electronically track posted transactions, mainly expenses, tasks, and invoices. Use the `AccountService` class to access accounts in the API.

The API provides the following model interfaces for accounts:

- The `Account` class, which contains information about the account.
- The `AccountInvolvedType` class, which contains the enums for posting an account to a contact.
- The `AccountOverdraftType` class, which contains the enums for how the account handles overdraft amounts.
- The `AccountPostingCriteria` class, which contains information about the account and its posting criteria.
- The `AccountPostingStatus` class, which contains the enums that indicate whether an expense, invoice, or task is posted to an account.
- The `AccountProjectType` class, which contains the enums for posting a project to an account.
- The `AccountTransaction` class, which contains information about an account transaction.
- The `AccountVendorType` class, which contains the enums for posting a vendor to an account.

Refer to the AccountService samples for examples of working with accounts in the API.

Use `AccountService` when working with account records.

Whenever a Service operation requires `AccountService`, set up `AccountService` using the following code:

```
AccountService accountService = platform.getAccountService();
```

Creating an Account

The following sample creates an account record. During account creation, you can set objects in the account record, such as the account name.

```
Account newAccount = accountService.newAccount("name", allocationLimit,
startOnDate, endOnDate);
```

Creating a Child Account

The following sample creates a child account record. During child account creation, you can use API interface methods to set objects in the account, such as the parent account.

```
Account newChildAccount = accountService.newAccount("name",
allocationLimit, startOnDate, endOnDate);
newChildAccount.setParentAccount(parentAccount);
```

Reading an Account

The following sample reads an account record. When you read an account, you retrieve the account so that you can make changes to it, such as activating or transferring money from it.

```
//The id initialized here is the id of an existing account
Long id = 1234567890L;
Account account = accountService.read(id);
```

Activating an Account

The following sample activates an account record.

```
accountService.activateAccount(accountToBeActivated);
```

Deactivating an Account

The following sample deactivates an account record.

```
accountService.deactivateAccount(account);
```

Allocating Money to an Account

The following sample allocates money to an account record. When you allocate money to an account, a parent account must already exist.

```
//Allocating money to an account - $5000
accountService.allocateMoney(account, amount, "description");
```

Transferring Money between Accounts

The following sample transfers money between two accounts. When you transfer money, a parent and child account must already exist.

```
accountService.transferMoney(accountFrom, accountTo, amount, "description");
```

Withdrawing money from an Account

The following sample withdraws money from an account record.

```
//Withdrawing from an account - $5000
accountService.withdrawMoney(account, new BigDecimal("5000"), "Withdraw to
zero");
```

Deleting an Account

The following sample deletes an account record. When you delete an account, the account must not have a balance.

```
//Account cannot have a balance before deletion. Make necessary withdrawals.
accountService.withdrawMoney(parentAccount, new BigDecimal("1000"), "Withdraw to
zero");
accountService.withdrawMoney(childAccount, new BigDecimal("234"), "Withdraw to
zero");


//When deleting a parent account with attached child accounts, the child
accounts also cannot have a balance. The deletion of a parent will delete its
child accounts as well.
accountService.delete(parentAccount);
```

1.4.2.5.1.2 Appointments

Appointments are scheduled events or meetings with a date, time, and attendees. Use the `AppointmentService` class to access appointments in the API.

The API provides the following model interfaces for appointments:

- The `Appointment` class, which contains information about an appointment.
- The `AppointmentAttendee` class, which contains information about an appointment attendee user and type.
- The `AttendanceType` class, which contains the enums that indicate if a contact will attend an appointment.

Refer to the AppointmentService samples for examples of working with appointments in the API.

Use `AppointmentService` when working with account records.

Whenever a Service operation requires `AppointmentService`, set up `AppointmentService` using the following code:

```
AppointmentService appointmentService = platform.getAppointmentService();
```

Creating an Appointment

The following sample creates an appointment. During appointment creation, you can use API interface methods to set objects in the appointment, including the start date, end date, and category.

```
// Retrieve the appointment service from the platform provided by the API
AppointmentService appointmentService = platform.getAppointmentService();

// Set up the initial required arguments to create a valid appointment
long fourDaysInMilliseconds = 86400000 * 4;
Date startOn = new Date();
Date endOn = new Date(startOn.getTime() + fourDaysInMilliseconds);

// Create a new appointment, passing in the arguments created above.
Appointment appointment = appointmentService("Appointment Subject", startOn,
endOn);

// Any non-required fields on the appointment may be set or added now
appointment.setAllDay(true);
appointment.addCategory("APPT_CATE_CUST");
```

Updating an Appointment

The following sample updates an appointment. When you update an appointment, you can use API interface methods to set objects in the appointment, including the primary key, new attendees, and custom fields.

```
// Retrieve the appointment service from the platform provided by the API.
AppointmentService appointmentService = platform.getAppointmentService();

// Retrieve the desired appointment.
long primaryKeyForAppointment = 2;
Appointment appointment = appointmentService.read(primaryKeyForAppointment);

// Modify fields.
appointment.addAttendee(platform.getUserService().findUserByUsername("TestUser"
), AttendanceType.WILL_ATTEND);
appointment.setBooleanFieldValue("customBooleanField", false);
```

Deleting an Appointment

The following sample deletes an appointment.

```
// Retrieve the appointment service from the platform provided by the API.
```

```
AppointmentService appointmentService = platform.getAppointmentService();

// An appointment can be deleted simply by passing in the primary key.
long primaryKeyForAppointment = 8;
appointmentService.delete(primaryKeyForAppointment);

// Or, you can pass in the appointment object.
Appointment appointment = appointmentService.read(primaryKeyForAppointment);
appointmentService.delete(appointment);

// In a rule, the delete will be committed when the rule completes.
// Otherwise, the delete will be committed immediately.
```

Searching for an Appointment

The following sample searches for all appointments in Room 101.

```
// Retrieve the appointment service from the platform provided by the API.
AppointmentService appointmentService = platform.getAppointmentService();

// Find all appointments located in Room 101
SearchCriteria criteria = new SearchCriteria(new
StringCriterion(Appointment.LOCATION).equalTo("Room 101"));

// Sort results by start date
SearchParameters parameters = new SearchParameters(new
SortField(Appointment.START_ON));

// Execute search and return results
return appointmentService.search(criteria, parameters);
```

1.4.2.5.1.3 Contacts

Contacts are people or companies for which your organization needs to store information, including employees, vendors, outside counsel, claimants, injured parties, witnesses, and agencies. Use the `ContactService` class to access contacts in the API.

The API provides the following model interfaces for contacts:

- The `Address` class, which contains information about a contact address.
- The `Company` class, which contains information that applies to only company contacts and extends the `Contact` class.
- The `Contact` class, which contains information that applies to all contacts.
- The `ContactRelation` class, which contains information about a contact relation.

- The `DefaultRate` class, which contains information about a default rate for a contact.
- The `Email` class, which contains information about a contact email address.
- The `FaxNumber` class, which contains information about a contact fax number.
- The `InternetAddress` class, which contains information about a contact web address.
- The `InvoiceTaskRate` class, which contains information about an invoice task rate for a contact.
- The `Person` class, which contains information that applies to only person contacts and extends the `Contact` class.
- The `PhoneNumber` class, which contains information about a contact phone number.
- The `Skill` class, which contains information about a contact skill.
- The `TaskRate` class, which contains information about a task rate for a contact.

Refer to the ContactService samples for examples of working with contacts in the API.

Use `ContactService` when working with contact records.

Whenever a Service operation requires `ContactService`, set up `ContactService` using the following code:

```
ContactService contactService = platform.getContactService();
```

Creating a Person Contact

The following sample creates a person contact record. During contact creation, you can use API interface methods to set objects in the contact record, including the contact address.

```
Person personContact = ContactService.newPerson("Bob", "Dole");
Address address = person.addAddress("ADDR_HOME");
address.setStreet("1234 Fake St.");
address.setCity("Austin");
address.setState("TX");
address.setPostalCode("78704");
address.setCountry("USA");
```

Creating a Company Contact

The following sample creates a company contact record. During contact creation, you can use API interface methods to set objects in the contact record, including the contact address and email.

```
Company companyContact = ContactService.newCompany("myLittleCompany");
Address address = company.addAddress("ADDR_BUS1");
address.setStreet("1234 Fake St.");
```

```
address.setCity("Austin");
address.setState("TX");
address.setPostalCode("78704");
address.setCountry("USA");
Email email = company.addEmail("MAIL_BUS1", "you@yourcompany.com");
```

Updating a Contact

The following sample updates a contact record. When you update a contact, you can use API interface methods to update objects in the contact record, including the contact fax number, email, and web address.

```
//Adding or updating a business fax number
FaxNumber faxNumber = person.addFaxNumber("FAXX_BUS1", "5123827322");

//Adding or updating a personal email
Email email = person.addEmail("MAIL_PER1", "you@yourcompany.com");

//Adding or updating a mobile phone number
PhoneNumber phoneNumber = person.addPhoneNumber("PHON_MOBI", "5123827322");

//Adding or updating a business web address
InternetAddress internetAddress = person.addInternetAddress("INET_BWEB",
"www.mitratech.com");

//Adding or updating a default skill
Skill skill = person.addSkill("SKIL_DEFA", 2);
```

Reading a Contact

The following sample reads a contact record. When you read a contact, you retrieve the contact so that you can make changes to it, such as updating or deleting it.

```
//The id initialized here is the id of an existing person contact
long personId = 49L;
Person personContact = ContactService.readPerson(personId);
//The id initialized here is the id of an existing company contact
long companyId= 23L;
Company companyContact = ContactService.readCompany(CompanyId);
```

Deleting a Contact

The following sample deletes a contact record. When you delete a contact, if the contact is associated with a user or any other object, remove the association first.

```
ContactService.delete(personContact);
ContactService.delete(companyContact);
```

1.4.2.5.1.4 Documents

In addition to the **Documents** tab where you can save any document, all records have a **Documents** page where you can save documents to that record. When working with documents in the API, you have the ability to perform the same functionality that you can use with documents in TeamConnect. Use the `DocumentService` class to access with documents in the API.

The API provides the following model interfaces for documents:

- The `Document` class, which contains information about a document. A document is anything that can appear in the documents list.
- The `Folder` class, which contains information about a document folder.
- The `Hyperlink` class, which contains a hyperlink.
- The `Shortcut` class, which contains a shortcut.
- The `File` class, which contains information about a file.
- The `FileContentType` class, which contains information about the file type of a document.
- The `FileVersion` class, which contains information for a file version.

Refer to the DocumentService samples for examples of working with documents in the API.

Use `DocumentService` when working with document records.

Whenever a Service operation requires `DocumentService`, set up `DocumentService` using the following code:

```
DocumentService documentService = platform.getDocumentService();
```

Creating a Document Record

The following sample creates a document. During document creation, you can add the document to a folder, create a shortcut for the folder, and check in the file, among other things.

```
DocumentService documentService = platform.getDocumentService();

// Get attachment folder of a record
Folder attachmentFolder =
documentService.getAttachmentFolderForDocumentOwner(getRecord());

// Create a folder under the attachment folder
Folder folder = attachmentFolder.addFolder("New Folder");
```

```
// Create a file directly under the attachment folder
FileContentType contentType =
platform.getLookupItemService().getFileContentTypeByFileExtension("txt");
byte[] fileContent = new byte[0];
File file = attachmentFolder.addFile("New File", fileContent, contentType);

// Create a hyperlink
Hyperlink hyperlink = folder.addHyperlink("Mitratech", "http://
www.mitratech.com");

// Create a shortcut to a folder
Shortcut shortcutToFolder = folder.addShortcut(folder);

// Create a shortcut to a file
Shortcut shortcutToFile = folder.addShortcut(file);

//Checkout file
documentService.checkOut(file);

//Checkin file
documentService.checkIn(file, fileContent, "check in comment");
```

Reading a Document Using its Path

The following sample returns a document using its folder path.

```
// Returns a document from the repository using its folder path
Document document = documentService.findDocumentByPath("path");
```

Retrieving the Root Folder

The following sample returns the root document folder.

```
// Returns the root document folder
Folder rootFolder = documentService.getRootFolder();
```

Copying a Document

The following sample copies a document to a new folder.

```
// Returns an exact duplicate of a given entity object
Document duplicateDocument = documentService.copyEntity(originalDocument);
```

```
// Copies a document to a new parent folder
documentService.copyDocument(document, parentFolder);
```

Moving a Document

The following sample moves a document to a new folder.

```
// Changes the location of a file
documentService.moveDocument(document, parentFolder);
```

Checking Out a Document

The following sample checks out a file and locks it so that other users cannot check it out.

```
// Checks out and locks a file for a single user to update
documentService.checkOut(file);
```

Undoing a Document Checkout

The following sample reverses a document checkout. The document remains unchanged and has no version number.

```
// Reverses the checkout of a file.
documentService.undoCheckOut(file);
```

Checking in a Document

The following sample checks in a document by converting the existing file to the new file version and replacing the file contents the byte array of the new contents.

```
// Checks in a file.
documentService.checkIn(file, data, "versionText");
```

Reverting a Document to a Previous Version

The following sample reverts a document to a previous version by converting the existing file to a different file version and replacing the file contents with the byte array of the new contents.

```
// Revert a document to a prior version
FileVersion version = documentService.getFileVersions(file).get(0);
documentService.revertTo(version);
```

1.4.2.5.1.5 Expenses

Expenses are internal costs for your business as a whole or that you can associate with an account. Use the `ExpenseService` class to access expenses in the API.

The API provides the `Expense` model interfaces for information about an expense.

Refer to the ExpenseService samples for examples of working with expenses in the API.

Use `ExpenseService` when working with expense records.

Whenever a Service operation requires `ExpenseService`, set up `ExpenseService` using the following code:

```
ExpenseService expenseService = platform.getExpenseService();
```

Creating an Expense

The following sample creates an expense record. During expense creation, you can use API interface methods to set objects in the expense record, including the user who is responsible for the expense and a description of the expense.

```
//Expense description is a mandatory field
Expense expense = expenseService.newExpense("Brand New Test Description");
User user = platform.getUserService().findUserByUsername("Expenser");
expense.setExpensedBy(user);
//Date will be converted to midnight GMT, then back to the local date/time
relative to GMT midnight
Calendar cal = Calendar.getInstance();
cal.set(2014, 01, 30);
expense.setExpenseDate(new CalendarDate(cal));
```

Updating an Expense

The following sample updates an expense record. When you update an expense, you can use API interface methods to update objects in the expense, including the expense name.

```
expense.setShortDescription("Expense Description Update");
//Adding new unit price for expensed item ($150) and quantity of units (2)
expense.setUnitPrice(BigDecimal.valueOf(150));
expense.setQuantity(BigDecimal.valueOf(2));
```

Reading an Expense

The following sample reads an expense record. When you read an expense, you can retrieve the last saved version of the expense to post, delete, or do something else with the record.

```
//The expenseHandle here is a preexisting Expense object
Expense lastSavedExpense = expenseService.readLastSaved(expenseHandle);
```

Posting an Expense

The following sample posts an expense record. When you post an expense, you can use API interface methods to prepare an expense for posting.

```
//Ensure that these options are enabled in an account to post an expense
account.setAllowPosting(true);
account.getAccountPostingCriteria().setAllowExpense(true);
expenseService.postExpense(expense);
```

Voiding an Expense

The following sample voids an expense record.

```
//An expense must be posted to be voided
expenseService.voidExpense(expense);
```

Deleting an Expense

The following sample deletes an expense record.

```
expenseService.delete(expense);
```

Getting Transactions for an Expense

The following sample returns a list of transactions for an expense record.

```
List<AccountTransaction> transactions =
expenseService.getTransactionsForExpense(expense);
```

Getting Transactions for an Expense Associated with an Account

The following sample returns a list of transactions for an expense record that is associated with an account.

```
List<AccountTransaction> transactions =
expenseService.getTransactionsForExpense(account, expense);
```

Searching for Expenses

The following sample searches for expenses with a start date of January 2012.

```
Calendar cal = Calendar.getInstance();
cal.set(2012, 1, 1);
Date startDate = cal.getTime();

Calendar cal = Calendar.getInstance();
cal.set(2012, 1, 31);
Date endDate = cal.getTime();

DateCriterion dateCriterion = new
DateCriterion(Expense.EXPENSE_DATE).between(startDate, endDate);

List<Expense> expenses = platform.getExpenseService().search(new
SearchCriteria(dateCriterion));
```

1.4.2.5.1.6 Groups

In TeamConnect, a group exists independently of its group account record, meaning that if you need to get or set any information for a group, you must use the group account. All TeamConnect users have their own group accounts (instances of `GroupAccount`) that include information such as the group display name and description. Use the `GroupService` class to access with groups in the API.

The API provides the following model interfaces for groups:

- The `Group` class, which contains the display name of the group. You can use this object to return the group name.
- The `GroupAccount` class, which contains all the fields for the group account in TeamConnect. You can use this object to get or set information for a user.

Because every display name is unique, the group name connects a group to its group account. When you create a new group, TeamConnect generates a key that you can also use to identify the group. Use the `entity.getPrimaryKey` method if you want to identify the group with its key.

Because group rights determine the code you can run, it is sometimes necessary to check whether the current user is part of a group. You can get the current user using the `UtilityService.getCurrentUser()` method.

Refer to the GroupService samples for examples of working with groups in the API.

Use `GroupService` when working with group records.

Whenever a Service operation requires `GroupService`, set up `GroupService` using the following code:

```
GroupService groupService = platform.getGroupService();
```

Creating a Group

The following sample creates a group. To create a group, you need to create a group account.

During group creation, you can use API interface methods to set objects in the group, such as its description.

```
//create a new group account with specified unique name and display name.
GroupAccount groupAccount = groupService.newGroupAccount("GroupUniqueName",
"GroupDisplayName");
groupAccount.setDescription("Description");
```

Updating a Group

The following sample updates a group. When you update a group, you can use API interface methods to update the group account, such as its description.

```
//get group with specified name.
Group group = groupService.getGroupForName("GroupName");

//retrieve the group account from the group.
GroupAccount groupAccount = groupService.getGroupAccountForGroup(group);
groupAccount.setDescription("newDescription");
```

Reading a Group

The following sample reads a group. When you read a group, you retrieve the group account so that you can make changes to it, such as updating or deleting it.

```
//read group and user accounts.
GroupAccount groupAccount = groupService.read(1L);
UserAccount userAccount = userService.read(1L);

//get the group and user from their corresponding accounts.
Group group = groupService.getGroupForGroupAccount(groupAccount);
User user = userService.getUserForUserAccount(userAccount);

groupService.addUserToGroup(group, user);
```

Finding a Group

The following sample finds a particular group. You can use the unique to find the group.

```
//get group based on its unique key.
Group group = groupService.getGroupForKey("uniqueKey");
//get all groups.
List<Group> allGroups = groupService.getAllGroups();
//get all users for a group.
```

```
List<User> usersForGroup = groupService.getAllUsersForGroup(group);
//get all user accounts for group account.
GroupAccount groupAccount = groupService.read(1L);
List<UserAccount> userAccountsForGroupAccount =
groupService.getAllUserAccountsForGroupAccount(groupAccount);
//get all groups for a user.
User user = userService.findUserByUsername("username");
List<Group> groupsForUser = groupService.getGroupsForUser(user);
```

Deleting a Group

The following sample deletes a group by deleting the group account.

```
GroupAccount groupAccount = groupService.read(1L);
groupService.delete(groupAccount);
//same as groupService.delete(groupAccount.getPrimaryKey());;
```

1.4.2.5.1.7 History Entries

All records have a **History** or **Narratives** page, where you can save an entry with a general comment or information about an update to the record. When working with history entries in the API, you view current entries or create new entries. Use the `HistoryService` class to work with history entries in the API.

The API provides the `History` model interface for information about specific entries.

Refer to the HistoryService samples for examples of working with histories entries in the API.

Use `HistoryService` when working with history records.

Whenever a Service operation requires `HistoryService`, set up `HistoryService` using the following code:

```
HistoryService historyService = platform.getHistoryService();
```

Creating a History Entry

The following sample creates a history entry. During history creation, you can use API interface methods to set objects in the entry, such as the history description and the category.

```
// Creates an empty history object.
History newHistory = historyService.newHistory("history", parentEntity);
```

Reading a History Entry

The following sample returns a history entry using its primary key.

```
// Reads the Entity object based on the primary key provided
historyService.read(primaryKey);
```

Finding a History Entry

The following samples return all history entries for a particular entity and the most recent entry for an entity.

```
List<History> histories = historyService.getHistories(entity);
// Return most recent history with specified category and for a particular
entity
History lastHistory = historyService.getLastHistory(entity, "HIST_STAT");
```

Deleting a History Entry

The following samples delete a history entity.

```
// Deletes the Entity object
historyService.delete(history);
// Deletes the Entity object based on the primary key provided
historyService.delete(primaryKey);
```

1.4.2.5.1.8 Invoices

Invoices are bills that your organization receives from vendors and that you can post against accounts and projects. When working with invoices in the API, use the `InvoiceService` class.

The API provides the following model interfaces for invoices:

- The `Invoice` class, which contains information about the invoice.
- The `InvoiceAdjustment` class, which is the base class for all invoice adjustment classes and contains basic information about the adjustment.
- The `InvoiceAdjustmentTarget` class, which contains the enums that indicate the amount being adjusted for the invoice summary.
- The `InvoiceAdjustmentType` class, which contains the enums that indicate the type of invoice summary adjustment.
- The `InvoiceCategoryAdjustment` class, which contains information about the invoice category adjustment, such as the task or expense code.
- The `InvoicePostingStatus` class, which contains the enums that indicate the posting status of the invoice.
- The `InvoiceHeaderAdjustment` class, which contains information about whether fees, expenses, or the total amount is the target of the header adjustment.

- The `InvoiceTimekeeperAdjustment` class, which contains information about the timekeeper and the project associated with the adjustment.
- The `InvoiceType` class, which contains the enums that indicate the type of invoice.
- The `LineItem` class, which contains information about a line item.
- The `LineItemType` class, which contains the enums that indicate the type of line item.
- The `AdjustmentInformation` class, which contains information about the adjustments.
- The `AdjustmentMethod` class, which contains the enums that indicate the method for adjusting the amount.

Refer to the InvoiceService samples for examples of working with invoices in the API.

Use `InvoiceService` when working with invoice records.

Whenever a Service operation requires `InvoiceService`, set up `InvoiceService` using the following code:

```
InvoiceService invoiceService = platform.getInvoiceService();
```

Creating an Invoice with Line Items

The following sample creates an invoice and adds line items. During invoice creation, you can use API interface methods to set objects in the invoice record and line items, including the line item rate, quantity, discount, and total.

```
// vendor is an existing contact
Company vendor = platform.getContactService().readCompany(45L);

CalendarDate invoiceDate = new CalendarDate();
Invoice invoice = invoiceService.newInvoice("456", vendor, invoiceDate);
//the line item original rate.
BigDecimal originalRate = new BigDecimal("23.00");
//the line item original quantity.
BigDecimal originalQuantity = new BigDecimal("500");
//the line item original discount.
BigDecimal originalDiscount = new BigDecimal("17.56");
//the line item original total.
BigDecimal originalTotal = new BigDecimal("34805743570");

LineItem newLineItem = invoiceService.newExpenseLineItem(invoice, invoiceDate,
"LNI$_EXPS_OCEX_E100_E101", originalRate,
originalQuantity).setOriginalDiscount(originalDiscount).setOriginalTotal(origina
lTotal);
```

Updating and Adjusting an Invoice

The following sample updates and adjusts an invoice. When you update an invoice, you can use API interface methods to set objects in the invoice record, including the start date. You also use API interface methods to adjust an invoice, such as for reducing an amount to a new amount.

```
CalendarDate periodStartDate = new CalendarDate();
invoice.setPeriodStartDate(periodStartDate);
AdjustmentInformation information =
new AdjustmentInformation(AdjustmentMethod.REDUCE_BY_AMOUNT, new
BigDecimal("45.00"));
invoiceService.adjustInvoiceFees(invoice, information);
```

Adjusting Invoice Line Items

The following sample adjusts an invoice line item. When you adjust a line item, you can use API interface methods to adjust the line item, such as changing a line item amount to a new amount.

```
AdjustmentInformation information =
new AdjustmentInformation(AdjustmentMethod.NEW_AMOUNT, new BigDecimal("22.56");
invoiceService.adjustLineItemRate(lineItemToAdjust, information);
```

Reading an Invoice

The following sample reads an invoice record. When you read an invoice, you retrieve the invoice so that you can make changes to it, such as updating or posting to it.

```
// id of existing invoice.
long id = 4982340L;
Invoice invoice = invoiceService.read(id);
```

Searching for an Invoice

The following sample searches for an invoice record.

```
// Returns all invoices that have an invoice date within the last 10 days
SearchCriteria criteria = new SearchCriteria(new
RelativeDateCriterion(Invoice.INVOICE_DATE).withinLast(10));
List<Invoice> invoices = invoiceService.search(criteria);
```

Posting an Invoice

The following sample posts an invoice.

```
invoiceService.postInvoice(invoiceToBePosted);
```

Voiding an Invoice

The following sample voids an invoice.

```
invoiceService.voidInvoice(invoiceToBeVoided);
```

Deleting an Invoice

The following sample voids an invoice.

```
invoiceService.delete(invoiceToBeDeleted);
```

1.4.2.5.1.9 Tasks

Tasks are internal assignments that you can bill to accounts if necessary. When working with tasks in the API, use the `TaskService` class.

The API provides the following model interfaces for tasks:

- The `Task` class, which contains information about the task.
- The `TaskAssignee` class, which contains information about the user assigned to a task.
- The `TaskPriority` class, which contains the enums that indicate the priority of the task.
- The `TaskWorkStatus` class, which contains the enums that indicate the status of the task.

Refer to the TaskService samples for examples of working with tasks in the API.

Use `TaskService` when working with task records.

Whenever a Service operation requires `TaskService`, set up `TaskService` using the following code:

```
TaskService taskService = platform.getTaskService();
```

Creating a Task

The following sample creates a task record. During task creation, you can use API interface methods to set objects in the task record, including the current assignee, associated project, and task rate.

```
Project project = platform.getProjectService().read(0L);
User assignee = platform.getUserService().findUserByUsername("task assignee");
Task task = taskService.newTask("This is a new task", assignee);
task.setActualHours(BigDecimal.TEN);
task.setRateAmount(BigDecimal.ONE);
task.setNote("Note: This is a new task");
task.setProject(project);
```

Reading a Task

The following sample reads a task record. When you read a task, you retrieve the task so that you can make changes to it, such as updating or posting to it.

```
//the primary key of an existing task
long id = 12345678910L;
Task task = taskService.read(id);
```

Searching for Tasks

The following sample searches for tasks with a start date of January 2012.

```
Date startDate = new SimpleDateFormat("yyyy-MM-dd").parse("2012-01-01");
Date endDate = new SimpleDateFormat("yyyy-MM-dd").parse("2012-01-31");
DateCriterion dateCriterion = new DateCriterion(new FieldPath(Task.START_DATE));

List<Task> tasksStartingInJanuary2012 = taskService.search(new
SearchCriteria(dateCriterion.between(startDate, endDate)));
```

Posting a Task

The following sample posts a task record. When you post a task, you can use API interface methods to update the task at the same time.

```
//the primary key of an existing task
long id = 12345678910L;
Task task = taskService.read(id);
task.setCompletedDate(new CalendarDate());
task.setCompletedPercent(BigDecimal.valueOf(100));
taskService.postTask(task);
```

Voiding a Task

The following sample voids a task record.

```
//the primary key of an existing task
long id = 12345678910L;
Task task = taskService.read(id);
taskService.voidTask(task);
```

Reassigning a Task

The following sample reassigns a task to another user.

```
//the primary key of an existing task
```

```
long taskId = 12345678910L;
Task task = taskService.read(taskId);

//the primary key of an existing user
long userId = 12345678910L;
User user = platform.getUserService().findUserByUsername("new assignee");

taskService.reassign(task, user);
```

Deleting a Task

The following sample deletes a task record.

```
//the primary key of an existing task
long taskId = 12345678910L;
Task task = taskService.read(id);

taskService.delete(task);
```

Getting a List of Transactions for a Task

The following sample returns a list of transactions for a task record.

```
//the primary key of an existing task
long taskId = 12345678910L;
Task task = taskService.read(id);

List<AccountTransaction> transactions =
taskService.getTransactionsForTask(task);
```

Getting a List of Transactions for a Task associated with an Account

The following sample returns a list of transactions for a task record that is associated with an account.

```
//the primary key of an existing task
long taskId = 12345678910L;
Task task = taskService.read(id);

//the primary key of an existing account
long accountId = 12345678910L;
Account account = accountService.read(id);
```

```
List<AccountTransaction> transactions =
taskService.getTransactionsForTask(account, task);
```

1.4.2.5.1.10 Users

In TeamConnect, a user exists independently of its user account record, meaning that if you need to get or set any information for a user, you must use the user account. All TeamConnect users have their own user accounts (instances of `UserAccount`) that include information such as username, password, and account status. Each user account is also linked to a contact record (instance of `Contact`) through the `getContact()` method. Use the `UserService` class to access with users in the API.

The API provides the following model interfaces for users:

- The `User` class, which contains some read-only fields for the user. You can use this object to specify a user.
- The `UserAccount` class, which contains all the fields for the user account in TeamConnect. You can use this object to get or set information for a user.
- The `UserType` class, which contains the enums that indicate the type of user.

Because every username is unique, the username connects a user to its user account. When you create a new user, TeamConnect generates a key that you can also use to identify the user. Use the `entity.getPrimaryKey` method if you want to identify the user with its key.

Because you can change the user who is running the code, it is sometimes necessary to identify the current user. You can get the current user using the `UtilityService.getCurrentUser()` method.

Refer to the UserService samples for examples of working with users in the API.

Use `UserService` when working with user records.

Whenever a Service operation requires `UserService`, set up `UserService` using the following code:

```
UserService userService = platform.getUserService();
```

Creating a User

The following sample creates a user record. You can use the `newUser()` and `newUserAcccount()` methods to create a user account and its associated user, the `saveUserAccount()` method to save the user, and the `getUserForUserAccount()` method to extract the user from its account.

During user creation, you can use API interface methods to set objects in the user record, including the corresponding contact, username, and password. Before creating a user, a corresponding contact record must already exist in TeamConnect or you must create a new contact using ContactService. As shown in the sample, when creating a user record, you must also create the user account.

```
//read or create a contact
Contact contact = contactService.readPerson(1L);
```

```
//create a limited privilege user account.
UserAccount limitedUserAccount =
userService.newLimitedPrivilegeUserAccount("username", "password", contact,
true);
limitedUserAccount.setActive(true);
limitedUserAccount.setChangePasswordNextLogin(true);
limitedUserAccount.setShortDescription("Description for limited privilage user
account");
//get user from limited privilege user account.
User limitedUser = userService.getUserForUserAccount(limitedUserAccount);

//create a normal user account.
UserAccount normalUserAccount = userService.newNormalUserAccount("username",
"password", contact);
normalUserAccount.setActive(true);
normalUserAccount.setChangePasswordNextLogin(true);
normalUserAccount.setShortDescription("Description for normal user account");
//get user from normal user account.
User normalUser = userService.getUserForUserAccount(normalUserAccount);

//create a super user account.
UserAccount SuperPowerfulUserAccount =
userService.newSuperUserAccount("username", "password", contact);
SuperPowerfulUserAccount.setActive(true);
SuperPowerfulUserAccount.setChangePasswordNextLogin(true);
SuperPowerfulUserAccount.setShortDescription("Description for a super powerful
user account");
//get user from super user account.
User superUser = userService.getUserForUserAccount(SuperPowerfulUserAccount);
```

Updating a User

The following sample updates a user record. When you update a user, you can use API interface methods to update fields in the user record, including the username, password, and active setting.

```
//read or create a user account.
UserAccount userAccount = userService.read(1L);
userAccount.setUsername("newUsername");
userAccount.setPassword("newPassword");
userAccount.setActive(true);
```

Reading a User

The following sample reads a user record. When you read a user, you retrieve the user so that you can make changes to the user account, such as updating or deleting it.

```
//read a user account.
UserAccount userAccount = userService.read(1L);
//get the user from the user account.
User user = userService.getUserForUserAccount(userAccount);
```

Finding a User

The following sample find a particular user record. You can use a username to find the user.

```
User user = userService.findUserByUsername("username");
```

Deleting a User

The following sample deletes a user record.

```
UserAccount userAccount = userService.read(1L);
userService.delete(userAccount);
//same as userService.delete(userAccount.getPrimaryKey());
```

1.4.2.5.2 Custom Objects

Custom objects are objects not included with TeamConnect Enterprise. The solution developer typically creates custom objects or they are included as part of a TeamConnect module, such as TeamConnect Legal. Custom objects are also called projects or matters.

Custom objects in the API have a resource service class for projects and Involved parties. These classes use the API model interfaces to work with the system fields of custom objects. All system fields have get and set methods in their owner interfaces. You can also get or set custom fields on custom objects.

1.4.2.5.2.1 Projects

Projects refer to custom objects with information that meets the business requirements of your organization. When working with projects in the API, use the `ProjectService` class.

The API provides the following model interfaces for projects:

- The `Project` class, which contains information about the project.
- The `ProjectAssignee` class, which contains information about a user assignees to a project.
- The `ProjectRelation` class, which contains information about a contact related to the Involved party.

Refer to the ProjectService samples for examples of working with projects in the API.

Use `ProjectService` when working with project records.

Whenever a Service operation requires `ProjectService`, set up `ProjectService` using the following code:

```
ProjectService projectService = platform.getProjectService();
```

Adding an Embedded Project to a Parent Project

The following samples update a project record by adding an embedded project to the project. As shown in these samples, you can use `Project` methods to add the embedded project.

```
EmbeddedEntity embeddedProject = project.addEmbeddedEntity("unique code");
//or
EmbeddedEntity embeddedProject = project.addEmbeddedEntityWithName("unique
code", "name");
//or
EmbeddedEntity embeddedProject =
project.addEmbeddedEntityWithNumberString("unique code", "id number as a
string");
//or
EmbeddedEntity embeddedProject = project.addEmbeddedEntity("unique code",
"name", "id number as a string");
```

Changing a Project's Phase

The following sample changes the phase of a project and updates the record. While making these changes, you can use API interface methods to update objects in the project record, such as the project name.

```
project.setContact(contact);
project.setMainAssignee(assignee);
projectService.changePhaseWithUpdate(project, "phase code of the phase to change
to");
```

Reading Child Projects for a Record

The following sample returns the child projects of a parent project.

```
List<Project> childProjects = projectService.getChildProjects(project, "unique
code of the child Custom Object Definition");
```

Searching for a Project

The following samples return an existing project. The second sample includes a `ProjectService.search()` method. This `search()` method is different from the

`ResourceService.search()` methods because the `ProjectService.search()` method includes the unique code of an existing project as a parameter.

```
//Reading a project
//The id initialized here is the id of an existing project
Long id = 1234567890L;
Project project = projectService.read(id);

//Searching for a project
List<Project> search("unique code", "search view key");
```

Deleting a Project

The following sample deletes and existing project.

```
//The id initialized here is the id of an existing project
Long id = 1234567890L;
projectService.delete(id);
```

1.4.2.5.2.2 Involved Parties

Involved parties are project participants that may not be TeamConnect users. When working with Involved parties in the API, use the `InvolvedService` class.

The API provides the following model interfaces for Involved parties:

- The `Involved` class, which contains information about an Involved party.
- The `InvolvedRelation` class, which contains information.

Refer to the InvolvedService samples for examples of working with Involved parties in the API.

Use `InvolvedService` when working with Involved party records.

Whenever a Service operation requires `InvolvedService`, set up `InvolvedService` using the following code:

```
InvolvedService involvedService = platform.getInvolvedService();
```

Adding an Involved Party to a Project

The following sample adds a new Involved party to a project.

During Involved party creation, you can to set objects in the Involved party record, including the Involved party unique code and the associated project.

```
Involved newInvolved = involvedService.newInvolved(project, contact,
```

```
"INPA_EXPE");
```

Reading an Involved Party

The following sample returns an Involved party.

```
//The id initialized here is the id of an existing project
Long id = 1234567890L
Involved involved = involvedService.read(id);
```

Searching for an Involved Party

The following sample creates an Involved party record. The sample includes an `InvolvedService.search()` method. This `search()` method is different from the `ResourceService.search()` methods because the `InvolvedService.search()` method includes the unique code of an existing project as a parameter.

```
List<Involved> involveds = involvedService.search("unique code", "search view
key");
```

Deleting an Involved Party

The following samples delete an Involved party record.

```
involvedService.delete(involved);
//or
//The id initialized here is the id of an existing project
Long id = 1234567890L;
involvedService.delete(id);
```

1.4.2.5.3 Related Records

Any system or custom object record can have related records. For example, contact and appointment records can have documents or histories related to them. In addition, project records can have related appointments, documents, histories, accounts, tasks, expenses, and child project records.

Do not confuse related objects with sub-objects, such as a contact's addresses or assignees of a project. To retrieve sub-objects, you must use the appropriate methods in the sub-object's owner interface.

The methods that you use to get related records depend on the types of related records. Projects have some methods for obtaining related records. In certain cases, to retrieve some related records, you must perform a search. Search for the following types of related records:

- The following related records of system objects and projects: tasks, accounts, appointments, expenses, invoices, history entries, and documents.

- Child accounts. While you can access the parent account from the child using `Account.getParentAccount()`, access child accounts of a parent by searching.

1.4.2.5.3.1 Getting Related Records of Projects

When you edit a field with a related record in a project record, you retrieve that field using the `Project` interface. When you cannot edit a related record field in the project record, the `ProjectService` interface has methods for retrieving information related to a project. Use the following methods to obtain related records for a project:

- `ProjectService.getChildProjects()`—Retrieves all child projects records of a particular type related to a `Project` record.
- `ProjectService.getInvolved()`—Retrieves all Involved party records related to a `Project` record.
- `Project.getParentProjecct()`—Retrieves the parent project of a `Project` record.
- `Project.getEmbeddedProjects()`—Retrieves all embedded project records of a particular type related to a `Project` record.
- `Project.getRelations()`—Retrieves all projects related to a `Project` record. The API retrieves information about each relation using the `ProjectRelation` class.

Note: *In some cases, when you need to narrow down the list of related records you are looking for, you may need to search.*

For complete samples in Java, see the Looping through Related Records of Projects and Copying Values from Parent to Multiple Child Records samples.

1.4.2.5.3.2 Using Custom Fields and Primary Keys to Relate Records

Certain situations require you to uniquely identify a system object record, such as a task or history record, that does not have a relationship to the current object record. To work around this situation, you can add a custom field to a record and store the primary key of another record to create a relationship. When you need to retrieve a record, you can use the primary key in this field. Typically, users do not see or modify this field.

For example, you might have a rule that creates a history record when someone creates a particular task for a project. After the task is complete, another rule might retrieve the history record and update it. In this situation, the project's history and the task have no direct relationship. To create a relationship, you can use the first rule to create a custom field in the task object definition and store the primary key of the history record. When you need to retrieve that history record in the second rule, you can use the custom field value.

The following code snippet demonstrates how the first rule retrieves the primary key of the history record and stores it in the custom field of a task:

```
// Store the primary key of the history in the record's custom field
task.setNumberFieldValue("history", new BigDecimal(history.getPrimaryKey()));
```

The following code snippet demonstrates how a different rule retrieves the history record using its primary key in the custom field of a task record:

```
// Retrieving the primary key of the history from the record's custom field
Long primaryKey = record.getNumberFieldValue("history").longValue();

// Retrieve the related history record
History retrievedHistory = platform.getHistoryService().read(primaryKey);
```

1.4.2.5.4 Sub-Objects

Sub-objects contain information that exists only within a record. You cannot access the sub-objects outside that record. For example, categories, appointment attendees, contact addresses, invoice line items, and project assignees are all sub-objects.

Some sub-objects are also associated with the system lookup tables that define the individual types of sub-objects. For example, the **Fax Type** system lookup table defines the different types of fax numbers that you can add to a contact record.

Sub-Objects in the API

Each sub-object is part of a main object. You retrieve sub-objects using a get method in the main object class. For example, if you want to retrieve a list of attendees for an appointment, use the `Appointment.getAttendees` method. If you want to retrieve the phone numbers for a contact, use the `Contact.getPhoneNumbers` method.

You can also add and remove sub-objects using the corresponding methods in the owner interfaces. For example, if you want to add an attendee to an appointment, use the `Appointment.addAttendee` method. If you want to remove an attendee from an appointment, use the `Appointment.removeAttendee` method.

Sub-Object Classes

The following table includes all the frequently used sub-objects and their classes.

Sub-Objects of Main TeamConnect Objects

Main Object Class	Specific Sub-Object Classes
`Account`	(None)
`Appointment`	`AppointmentAttendee`

`Contact`	`Address`
	`Email`
	`FaxNumber`
	`InternetAddress`
	`PhoneNumber`
	`DefaultRate`
	`TaskRate`
	`InvoiceTaskRate`

	ContactRelation Skill
Document	(None)
Expense	(None)
History	(None)
Involved	InvolvedRelation
Invoice	LineItem
Project	ProjectAssignee Phase ProjectRelation
Task	TaskAssignee

1.4.2.5.4.1 Project and Task Assignees

Assignees are sub-objects of users assigned to a project or task. The methods for assignees in respective `Task` and `Project` interfaces reflect the differences between task and project assignees.

Project and Task Assignee Differences

Project Assignees	Task Assignees
Project assignees are instances of the `ProjectAssignee` interface.	Task assignees are instances of the `TaskAssignee` interface.
Projects can have multiple or no assignees. Projects with assignees always have one main assignee.	Tasks must have a current assignee and they can only have one assignee at a time. By default, the user who creates a task record is set as the assignee of that task.
A project has a list of assignees that includes the main assignee, other active assignees, and inactive (unassigned) assignees.	A task includes a list of all previously assigned users as well as the current assignee.
You can remove (or unassign) a project assignee without adding a new one.	When you reassign a task, it adds the new assignee as the current assignee and unassigns the previously set assignee.
Each project assignee has an assignee role.	Task assignees do not have roles.

Instances of `ProjectAssignee` or `TaskAssignee` are always associated with a user. When you add an assignee to a task or project, you must provide a user in order to create the assignee.

For example, the `Project.addAssignee()` method assigns a project to an existing user, as shown in the following sample, and returns `ProjectAssignee` with the assigned user.

```
project.addAssignee("ROLE_TREE_POSI", user);
```

1.4.2.5.4.2 Contact Sub-Objects

Contacts have many sub-objects, including the sub-objects with the following interfaces:

- `Address`
- `Email`
- `FaxNumber`
- `InternetAddress`
- `PhoneNumber`
- `DefaultRate`
- `TaskRate`
- `InvoiceTaskRate`
- `ContactRelation`
- `Skill`

In some sub-object lists for a contact, one of the list items is the primary item. Specific methods for getting and setting the primary sub-objects are available in the `Contact` interface. For example, the phone and fax sub-objects have a primary phone number and a primary fax number. You can retrieve these primary sub-objects with the `getPrimaryPhoneNumber()` and `getPrimaryFaxNumber()` methods.

All of the sub-objects of contact have types, which describe the values you enter. Most sub-objects store these types in the system lookup tables, with the exception of the rate sub-objects. The rate sub-objects (`DefaultRate`, `TaskRate`, and `InvoiceTaskRate`) are categories and instances of `CategoryDefintion`.

1.4.2.5.5 Categories

Categories have the following two purposes:

- They provide a simple way to organize and search for records.
- They organize custom fields by associating each custom field with a category. For custom fields to appear at all times, you can associate them with the root category. Users can display or hide custom fields by selecting or deselecting the associated categories.

In every object definition, the starting point for the category hierarchy is the root category. The hierarchy also includes a primary category, which the user sets manually. Upon record creation, the root category is set as the primary category.

Categories in the API

In the API, you represent categories with the following interfaces:

- `CategoryService`—The general category class that you use to obtain information about the categories for an object definitions or a tree position.

  Note: *Use the following code to access CategoryService:* `CategoryService categoryService = platform.getCategoryService();`

- `Category`—The class that you use to obtain information about added categories and their hierarchical positions in a record, including any children and parent categories. A `Category` is an instance of a `CategoryDefinition`.
- `CategoryDefinition`—The class that you use to obtain information about categories that exist for an object and their associated custom fields.
- `EnterpriseEntity`—The class that includes methods that system objects and other interfaces extend. This class includes the `getAllCategories()`, `getPrimaryCategory()`, `hasCategory()`, and `setPrimaryCategory()` methods.
- Object model interfaces with categories—These class also have methods for getting and setting categories. For example, the `Invoice` class has the following methods: `addCategory()`, `getCategories()`, and `removeCategory()`. Other object model interfaces may have different category methods.

Note: *In instances of an Involved object, categories are known as "involved roles."*

Category Tree Positions

A full tree position uniquely identifies a category. The root category's full tree position is the same as the unique code of the object definition. The full tree position of non-root categories is a combination of the following with underscores as delimiters:

- The full tree position of the category's parent category
- The partial tree position of the category itself

For example, category B with partial tree position BBBB is under category A with partial tree position AAAA, which is under the root category of the Appointment object definition with the unique code APPT. As a result, the full tree position of category B is APPT_AAAA_BBBB, and the full tree position of category A is APPT_AAAA.

1.4.2.5.5.1 Adding or Deleting Categories in Records

If you want to select a category for a record, you can use the `addCategory()` method from the object's model class. The following code sets External, with a tree position of CONT_EXTE, as a category for a contact:

```
contact.addCategory("CONT_EXTE");
```

If you want to deselect a category from a record, you can use the `removeCategory()` method from the object's model class. The following code removes External from the list of selected categories for a contact:

```
contact.removeCategory("CONT_EXTE");
```

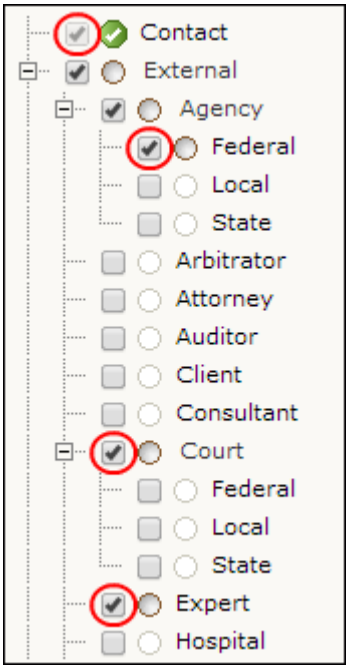
For a sample, see Updating Added Categories.

1.4.2.5.5.2 Returning Categories for a Record

If you want to know the categories that are selected for a record, you can use the `EnterpriseEntity.getAllCategories()` method. The following code returns all selected categories, including the parent categories, for a contact:

```
List<Category> allCategories = contact.getAllCategories();
```

If you want to know the lowest-level categories that are selected for a record, you can use the `getCategories()` method from the object's model class. The lowest-level categories include all the categories that do not have any child categories. If a selected category has lower-level categories that are not selected, `getCategories()` only returns the selected category. For example, the circled categories in the following image are the lowest-level selected categories.



Categories

If the contact record has the categories selected in the previous image, the following code returns the **Contact**, **Federal**, **Court**, and **Expert** categories for the contact:

```
List<Categories> categories = contact.getCategories();
```

In this example, the code does not return **Agency** because **Federal** is a lower level category, and it returns **Court** because none of the lower level categories for **Court** are selected.

1.4.2.5.5.3 Getting and Setting Primary Categories

If you want to set the primary category of an object, you can use the `EnterpriseEntity.setPrimaryCategory()` method. The following code sample provides an example of how to create a new contact and set its primary category:

```
// Retrieve the contact service
ContactService contactService = platform.getContactService();

// Create a new, blank contact. The vendor record is a contact which has
// previously been created. Set some categories for the contact record
Contact contact = contactService.newCompany("vendor");
contact.setPrimaryCategory("CONT_EXTE_LAWF");
contact.addCategory("CONT_EXTE_VEND");

// Save the contact to the database
contactService.create(contact);
```

If you want to return the primary category of an object, you can use the `EnterpriseEntity.getPrimaryCategory()` method. The following code sample returns the primary category of a contact:

```
Category primaryCategory = contact.getPrimaryCategory();
```

For a complete sample, see the Checking the Default Role in Related Involved Records sample.

1.4.2.5.5.4 Checking Records for Specific Categories

To return whether a record has a particular category, you can use the `hasCategory()` method and the full tree position of the category. The following code sample checks if a contact has the "Vendor" category:

```
boolean isVendor = contact.hasCategory("CONT_EXTE_VEND");
```

For a sample, see the Checking Lists of Added Categories sample.

1.4.2.5.5.5 Custom Fields

Custom fields exist for capturing your organization's custom information in TeamConnect. You create custom fields within object definitions under categories in an object. Within records in TeamConnect, users make custom fields available by selecting the categories with that custom field. Custom fields that appear at all times are under the root category.

In the TeamConnect API, the following interfaces represent custom field information:

- `EnterpriseEntity` and `Project`—You use methods in these classes to get and set custom fields, depending on the custom field type.
- `CustomField`—You use this class to retrieve information about a custom field. You mainly use this custom field and its subclasses when constructing search criteria.

You can get and set most custom fields using methods in the `EnterpriseEntity` interface. The `Project` interface has methods to get and set custom fields of Involved type.

To get or set a value in a custom field, you need the following information to locate the necessary field:

- (If the field is not under a root category) The full tree position of the category for the field.

  You can find the full tree position of a category in the UI of the TeamConnect Setup.
- The name of the custom field, not its label.

  You can find field names on the **Custom Fields** tab of the corresponding object definition in the TeamConnect user interface.
- The type of custom field.

  Depending on the field type, you use different methods to get or set the value in the custom field.
- (If the custom field is of type List) The full tree position of the lookup table item in the list.

The following table lists all the methods from the `EnterpriseEntity` and `Project` interfaces that allow you to get or set custom field values according to field type.

Custom Field Types and Method Names

Field Type	get and set Methods	Class(es) with Methods
Check Box (Boolean)	`getBooleanFieldValue()` `setBooleanFieldValue()`	`EnterpriseEntity`
Date	`getDateFieldValue()` `setDateFieldValue()`	`EnterpriseEntity` `CalendarDate`
Involved	`getInvolvedFieldValue()` `setInvolvedFieldValue()`	`Project`
List	`getLookupFieldValue()` `setLookupFieldValue()`	`EnterpriseEntity`
Memo (Text)	`getMemoFieldValue()` `setMemoFieldValue()`	`EnterpriseEntity`
Number	`getNumberFieldValue()` `setNumberFieldValue()`	`EnterpriseEntity`
Custom Object	`getProjectFieldValue()` `setProjectFieldValue()`	`EnterpriseEntity`

Text	`getTextFieldValue()` `setTextFieldValue()`	`EnterpriseEntity`

You can retrieve custom fields from any of the objects that extend `EnterpriseEntity`, as shown in the following code:

```
record.getCalendarDateFieldValue("fieldName");
record.getLookupFieldValue("fieldName");
```

If a custom field is part of a non-Root category, you must include the tree position when retrieving the value of the custom field:

```
record.getBooleanFieldValue("TREE_POSI", "fieldName");
record.getLookupFieldValue("TREE_POSI", "fieldName");
```

In the following code sample, you can get the value of the **LossAmount** custom field of type Number, which has a category with the full tree position CLAM_FEAT:

```
record.getNumberFieldValue("LossAmount", "CLAM_FEAT");
```

For a complete sample, see the Comparing Custom Field Qualifiers sample.

You can set different kinds of values for custom fields of objects that extend `EnterpriseEntity`. To set a value in a list custom field, you can enter a field name and the value using `EnterpriseEntity` methods, as shown in the following code:

```
record.setBooleanFieldValue(fieldName, true);
record.setLookupFieldValue(categoryTreePosition, fieldName, "ITEM");
```

In the following code sample, you can set the value of the memo field **OtherDistractions** and use its category full tree position of CLAM_FEAT:

```
record.setMemoFieldValue("CLAM_FEAT", "OtherDistractions", "Claimant was reading
a map.");
```

For a complete sample, see the Checking and Setting Values in Custom Fields of Type List sample.

1.4.2.5.6 Lookup Table Fields

Lookup tables provide options in List fields of TeamConnect. Each option in a List field is a lookup table item. TeamConnect has two types of lookup tables:

- System lookup tables that TeamConnect includes by default. You can modify them by adding and removing items, but you cannot delete them.

- Custom lookup tables that you can create as custom fields. Unlike system lookup tables, you can delete custom lookup tables. However, determine whether other design components are dependent on the custom lookup table before deleting it.

Because items in a lookup table have hierarchical relationships between them, you can use full tree positions to uniquely identify all lookup table items. Tree positions for system lookup tables and custom lookup tables are different.

Lookup Tables in the API

In the API, you represent lookup tables and their items with the following interfaces:

- `LookupItemService`—Includes methods that you use to return information about lookup tables and items. For example, you can use the `getCustomLookupItems()` method to retrieve a list of items in a custom lookup table:

```
lookupItemService.getCustomLookupItems("CODE");
```

Note: *Use the following code to access LookupItemService:* `LookupItemService lookupItemService = platform.getLookupItemService();`

- `LookupItem`—Includes methods for retrieving information about lookup table items. For example, the following code uses the `getStoredValue()` method to retrieve the stored value of a lookup table item:

```
contact.getLookupFieldValue("categoryFullTreePosition",
"fieldName").getStoredValue();
```

- `EnterpriseEntity`—Includes methods that system objects and other interfaces extend. For example, you can use the `getLookupFieldValue()` method with the `Contact` model interface to return the value for a custom lookup field:

```
contact.getLookupFieldValue(String treePosition, String fieldName);
```

LookupItemService Interface

Most `LookupItemService` methods return information about the following types of lookup tables and lookup table items:

- System lookup tables—You can retrieve all system lookup tables. Each system lookup table in TeamConnect has a LookupItemService method. For example, use `lookupItemService.getContactPhoneNumberTypes()` to return a list of the lookup table items for the phone number field in a contact.
- Custom lookup tables—You can retrieve a custom lookup table and item. `getCustomLookupItemForTreePostion()` returns one item in a lookup table, and `getCustomLookupItems()` returns the list of items in a custom lookup table.
- Currency—You can retrieve and update the currency lookup table. `getCurrencyItemList()` returns the list of currencies in the lookup table, and `updateCurrencyRates()` adds a currency entry to the lookup table.

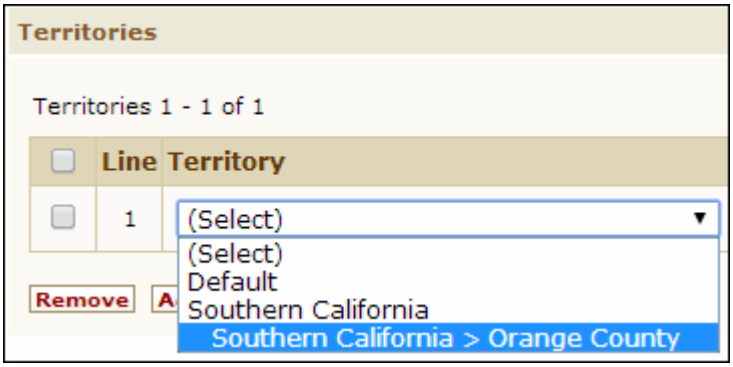
- Expenses—You can retrieve information about an expense category. `readLineItemExpenseCode()` returns an item in the expense category lookup table, and `readLineItemExpenseUnitPrice()` and `readExpenseRateScaleValue()` return rates and units associated with a category.
- Document types—You can retrieve document types. For example, `getAllDocumentTypes()` returns the list of document types in a lookup table.

1.4.2.5.6.1 System Lookup Table Tree Positions

The full tree position of a system lookup table item is a combination of the following codes with underscores as delimiters:

- The full tree position of the lookup item's parent item.
- The partial tree position of the item.

In the following image, the **Territory Type** lookup table in contact records has the code of TERR. If **Orange County** has the partial tree position OCNT and **Southern California** has the partial tree position SOCA, the full tree position of Orange County is TERR_SOCA_OCNT.



Territory System Lookup Table

The following code samples provide examples of how you can use the tree position to retrieve a system lookup table item.

```
LookupItemService lookupItemService = platform.getLookupItemService();

// Retrieving the "Apple" item within the Fruits custom table
CustomLookupItem apple =
lookupItemService.getCustomLookupItemForTreePosition("FRUI_APPL");

// Retrieve all items in the Fruits custom table
List<CustomLookupItem> fruits = lookupItemService.getCustomLookupItems("FRUI");

// Retrieving the "Mobile" item within the Phone Type system table;
SystemLookupItem mobile =
lookupItemService.getSystemLookupItemForTreePosition("PHON_MOBI");
```

The following table includes all system lookup tables, their associated `LookupItemService` method, and their unique codes. Use the unique code of each system lookup table at the beginning of the full

tree position for a lookup table item. The table also lists API model interfaces associated with the lookup tables. If the lookup tables are not associated with a string field, the API class column does not apply.

System Lookup Table Names

Friendly name	Associated LookupItemService method	Associated API class	Unique code
Activity Item	`getActivityItems()`	N/A	`ACTI`
Address Type	`getContactAddressTypes()`	`Address`	`ADDR`
Contact Relation Type	`getContactRelationTypes()`	`ContactRelation`	`CONR`
Country Item	`getCountries()`	N/A	`COUN`
Email Type	`getContactEmailTypes()`	`Email`	`MAIL`
Fax Type	`getContactFaxNumberTypes()`	`FaxNumber`	`FAXX`
Internet Address Type	`getContactInternetAddressTypes()`	`InternetAddress`	`INET`
Invoice Rejection Reason	`getInvoiceRejectionReason()`	N/A	`INRR`
Phone Type	`getContactPhoneNumberTypes()`	`PhoneNumber`	`PHON`
Project Assignee Type	`getProjectAssigneeRoles()`	`ProjectAssignee`	Unique code of the respective custom object definition.
Project Relation Type	`getProjectRelationTypes()`	`ProjectRelation`	`PRJR`
Resource Type	`getAppointmentResourceTypes()`	N/A	`RESO`
Skill Type	`getContactSkillTypes()`	`Skill`	`SKIL`
Territory Type	`getContactTerritoryTypes()`	N/A	`TERR`
State Item	`getStatesForCountryCode()`	N/A	`STAT`

1.4.2.5.6.2 Custom Lookup Table Tree Positions

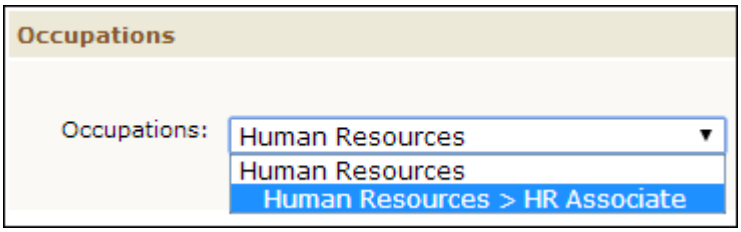
When you create a custom lookup table, you specify the unique code of the lookup table in the form of a 4-character alphanumeric combination. For example, the unique code of the **Occupations** custom lookup table might be OCCU.

The full tree position of the root node includes the unique code of the custom lookup table and ROOT. For example, the full tree position of the root node of the **Occupations** lookup table is OCCU_ROOT.

The full tree position of non-root custom lookup table items include the following with underscores as delimiters:

- The unique code of the custom lookup table.
- The partial tree position of the root node (ROOT).
- The partial tree position of any parent items.
- The partial tree position of the item.

For example, the **Occupations** custom lookup table in the following image has an **HR Associate** item with a partial tree position of HRAS and a **Human Resources** item with the partial tree position HURE. As a result, the full tree position of the **HR Associate** item is OCCU_ROOT_HURE_HRAS.



Occupations Custom Lookup Table

The following code samples provide examples of how you can use the tree position to retrieve a custom lookup table item.

```
CustomLookupItem customItem =
lookupItemService.getCustomLookupItemForTreePosition("treePosition");


//Retrieving "Big Budget Hits" custom item in the "Released Movies" custom table
//Table Code: RMOV, Item Tree Position: BBHT
CustomLookupItem movie =
lookupItemService.getCustomLookupItemForTreePosition("RMOV_ROOT_BBHT");


//Retrieving "Comedy" child item of the "Big Budget Hits" custom item
//Table Code: RMOV, Item Tree Position: CMDY
CustomLookupItem comedy =
lookupItemService.getCustomLookupItemForTreePosition("RMOV_ROOT_BBHT_CMDY");
```

1.4.2.5.6.3 Getting or Setting Values in List Custom Fields

The following sections demonstrate how to get or set values in List custom fields.

Getting Values in Lists

The `EnterpriseEntity.getLookupFieldValue()` method retrieves the item selected for a List custom field. You can use this value, which is a `LookupItem` object, to retrieve the full tree position of the lookup item. The following code provides an example of how you can retrieve the full tree position of a List custom field in the contact object:

```
contact.getLookupFieldValue("APPT_AAAA_BBBB", "HR Associate").getKey();
```

In addition to contacts, you can use this code for any objects that extend `EnterpriseEntity`, such as projects or expenses.

Setting Values in Lists

When you set values in List custom fields, you must include the following information:

- The full tree position of the category for the custom field.
- The name of the List custom field.
- The full tree position of the item in the lookup table.

You can use the `EnterpriseEntity.getLookupFieldValue()` without the full tree position of the category, as shown in the following code sample.

```
contact.setLookupFieldValue("HR Associate", "OCCU_ROOT_HURE_HRAS");
```

If you know the tree position, you can also use it as one of the parameters for the method:

```
contact.setLookupFieldValue("APPT_AAAA_BBBB", "HR Associate",
"OCCU_ROOT_HURE_HRAS");
```

For a complete sample of custom list fields in Java, see Checking and Setting Values in Custom Fields of Type List.

1.4.2.6 Common API Functions

If you need to perform common functions using the API, you can use the `UtilityService` interface. This interface encompasses miscellaneous methods that you might use throughout your code. These methods do not set anything in TeamConnect, but instead they retrieve information or run your code as a different user.

Note: *Use the following code to access UtilityService:* `UtilityService utilityService = platform.getUtilityService();`

`UtilityService` includes methods for the following types of actions:

- *Converting or changing formats*—For example, you can use the `booleanValueForString()` method to convert a string to a boolean and the `formatDateByUserSetting()` method to format the date based on the user's date settings, as shown in the following code samples.

```
UtilityService utilityService = platform.getUtilityService();

//Converting strings to booleans and returning boolean true.
utilityService.booleanValueForString("YES");
utilityService.booleanValueForString("TRUE");
utilityService.booleanValueForString("1");

//Converting strings to booleans and returning boolean false.
utilityService.booleanValueForString("NO");
utilityService.booleanValueForString("FALSE");
utilityService.booleanValueForString("0");

//Formatting the date for the user based on the current settings for that
User user = utilityService.getCurrentUser();
Date today = new Date();
String formattedDate = utilityService.formatDateByUserSetting(today, user);
```

- *Retrieving information based on the current day, time, or user*—For example, you can also use the `getCurrentUser()` method to retrieve the user object for the user currently running the code. You can also use the `getBeginningOfToday()` method to return the start of the current day in the current user's time zone, as shown in the following code sample.

```
Date date = utilityService.getBeginningOfToday();
```

- *Retrieving information about single-project object definitions*—For example, you can use the `getSingletonForObjectDefinition()` method to return the single project for an object definition, as shown in the following code sample.

```
// Unique 4-letter code for custom project.
String projectDefinitionUniqueCode = "PROJ";
Project singletonProject =
utilityService.getSingletonForObjectDefinition(projectDefinitionUniqueCode);
```

- *Retrieving system information*—For example, you can use the `getDatabaseInfo()` to return information about the TeamConnect database, as shown in the following sample.

```
DatabaseInfo databaseInfo = utilityService.getDatabaseInfo();
String serverName = databaseInfo.getServerName();
String databaseName = databaseInfo.getDatabaseName();
Date databaseCreatedOn = databaseInfo.getDatabaseCreatedOn();
String version = databaseInfo.getVersion();
DatabaseProduct product = databaseInfo.getDatabaseProduct();
```

- *Comparing two different days*—For example, you can use the `isDayAfter()` method to determine whether one date occurs before or after another date, as shown in the following sample about comparing the created on date for two different invoices.

```
//find two invoices
Long invoiceId1 = 1L;
Invoice invoice1 = invoiceService.read(invoiceId1);
Long invoiceId2 = 2L;
Invoice invoice2 = invoiceService.read(invoiceId2);
//Determine if invoice2 was created after invoice1
if (utilityService.isDayAfter(invoice2.getCreatedOn(),
invoice1.getCreatedOn()))
{
    //do something
}
```

- *Sorting a list*—You can use the `sortUsingPath()` method to sort a list, as shown in the following sample.

```
List<User> users = platform.getUserService().getAllUsers();
//sort users based on the contact primary city. In ascending order.
FieldPath pathToContactsPrimaryCity = new
FieldPath(User.CONTACT).add(Contact.PRIMARY_ADDRESS).add(Address.CITY);
boolean isAscending = true;
List sortedUsers= platform.getUtilityService().sortUsingPath(users,
pathToContactsPrimaryCity, isAscending);
```

- *Running code as a different user*—You can use the `runAsSystemUser()` method to run code as the system user. You can also use the `runAsUser()` method to run code as a specific user, as shown in the following sample.

```
//get current user
User user = utilityService.getCurrentUser();
//run a call as the user
utilityService.runAsUser(user, new Callable() {
    @Override
    public void call() {
        //do something
    }
});
```

1.4.2.6.1 Running Code as the System User

Because every record is subject to security, change to a user you know has the appropriate rights to ensure that your code runs. To execute code regardless of your functional or record security rights, you can run code as if you are the system user.

Note: *If you want code to fail if the current user does not have the appropriate rights, you do not need to run code as the system user.*

The system user, which has the username "system" and contact name "system, system" includes the following characteristics:

- Has all rights at all times.
- Cannot be added to a group.
- Cannot be denied security rights in a record's **Security** tab.
- Can always execute any method in the API.

To run code as the system user inside a call, use `utilityService.runAsSystemUser()`.

`UtilityService` includes four `runAsSystemUser()` methods. The method you use depends on whether you want to return data and/or execute additional code if the call throws an exception.

The following code snippet indicates how to execute code as a system user without returning any results:

```
utilityService.runAsSystemUser(new Callable() {
    public void call() {
    // Place your code here.
    }
});
```

The following code snippet indicates how to execute code as a system user and return data:

```
String username = utilityService.runAsSystemUser(new
CallableWithReturn<String>() {
    public String call() {
        String answer = "";
        // Place your code here.
        return answer;
    }
});
```

The following code snippet indicates how to execute code as a system user without returning any results and also executing additional code if the call throws an exception:

```
try {
    utilityService.runAsSystemUser(new
    CallableWithException<FileNotFoundException> callable() {
        public void call() throws FileNotFoundException {
            throw new FileNotFoundException();
        }
    });
```

```
} catch (FileNotFoundException e) {
    System.out.println("Couldn't find file!");
}
```

You can combine the last two code snippets to execute code that returns data or that executes additional code in the case of an error.

1.4.2.6.2 Getting Current User Information

The current user is the user who is triggering a rule or invoking an action. When you want code to execute regardless of the current user's rights, you change the current user to the system user. Because the user running the code can change, you may need to retrieve the current user.

To retrieve the user object for the current user, use the `utilityService.getCurrentUser()` method.

The following code sample retrieves the username of the current user. This code is useful when the current user is a required condition. You can use the username to determine whether the current user meets the condition.

```
String username = UtilityService.getCurrentUser().getUsername();
```

The following code sample retrieves the contact record of the current user.

```
User currentUser = utilityService.getCurrentUser();
Contact userContact = userService.getUserAccountForUser(user).getContact();
```

The following code sample retrieves the groups to which the current user belongs.

```
utilityService.getCurrentUser.getGroupList();
```

The following code sample checks the list of groups for the current user to find out if the user is part of a specific group. You can use this code as a template to check other conditions in addition to the group membership of the current user.

```
// Initialize services.
UtilityService utilityService = platform.getUtilityService();
GroupService groupService = platform.getGroupService();

//get group list for user.
User currentUser = utilityService.getCurrentUser();
List<Group> groupsForUser = groupService.getGroupsForUser(currentUser);

//Check to see if specific group is part of the list.
for (Group group : groupsForUser) {
    if (group.getDisplayName().equals("specifiedGroupName")) {
        return false;
```

```
    }
}
return true;
```

1.4.2.7 Security Rights

TeamConnect includes the following types of security settings:

- Record security—The security settings of a record. You handle record security from the **Security** page in a record.
- Functional security—The rights of each user or group account to perform specific operations or functions on TeamConnect objects. Functional security affects all records of a specific object type.

Note: *Use the following code to access SecurityService:* `SecurityService SecurityService = platform.getSecurityService();`

1.4.2.7.1 Record Security

You can use record security in the API to update record permissions and determine whether a user or group can access a record. The `SecurityAccess` interface includes all the methods for retrieving and updating permissions for users and groups. To call these methods, use the `GroupSecurityAccess` and `UserSecurityAccess` interfaces, which extend `SecurityAccess`.

Updating the security of a record has two parts:

- Creating the security object—Before you can update the security of a record, you must create a security object for the user or group. This security object is specific to the record and the user or group. When you create this security object, you also allow or deny the user or group read access to the record. Create this object with the `allowUserAccess()`, `allowGroupAccess()`, `denyUserAccess()`, or `denyGroupAccess()` methods in the `EnterpriseEntity` interface.

  Note: *A record can only have one security object for each user or group. If you try to create a security object for a user or group that already exists, you receive an error.*

- Updating security—After you create the security object, you can use the `SecurityAccess` methods through the `UserSecurityAccess` and `GroupSecurityAccess` interfaces to update permissions. If you want to update a security object that already exists, use the `getUserSecurityAccessList()` or `getGroupSecurityAccessList()` methods in `EnterpriseEntity` to retrieve a record's security object from a list.

Allowing Security Access

To allow access to a record, create the security object and give the user or group read access.

For example, if you are adding group rights to a record, you use the `EnterpriseEntity.allowGroupAccess()` method to create a security object and grant read access to the group. After you retrieve the group, you can use methods part of the `GroupSecurityAccess` interface to update permissions, as shown in the following code sample.

```
public void allowGroupSecurityAccessObject(Contact record) {
    // Creates a record-level security object which (by default) grants the
    group read access to the record
    Group group = platform.getGroupService().getGroupForName("test group");
    GroupSecurityAccess gsa = record.allowGroupAccess(group);

    // To allow more permissions, specify them individually
    gsa.addUpdate().addDelete().addChangeSecurityAccess();
}
```

If group rights for a record already exist, you cannot use the `allowGroupAcess()` method to retrieve the security object. If you want to update the group rights of the record, you must retrieve it from the list of records for the group's security objects using the `getGroupSecurityAccessList()` method.

Denying Security Access

To deny access to a record, create the security object and deny the user or group specific rights.

For example, if you want to deny group access to a record, you must create the security object using the `EnterpriseEntity.denyGroupSecurityAccess()` method. In addition, when you create the group, you can specify which security permissions you want to deny for the group's access to that record, as shown in the following code sample:

```
public void denyGroupSecurityAccessObject(Contact record) {
    //Creates a record-level security object which includes the specified
    permissions
    Group group = platform.getGroupService().getGroupForName("demo group");
    GroupSecurityAccess gsa = record.denyGroupAccess(group, true, true, true,
    true);

    // Check if the object contains the delete permission
    if (gsa.isDelete()) {
        record.setNote("Can be deleted by " + group.getDisplayName());
    }
}
```

Note: *Specifying true for the Boolean parameters of the denyGroupAccess() method denies access to a particular right, but specifying false does not allow or deny access.*

If you want to update a group's rights after creating the security object, you can use the `SecurityAccess` methods through the `GroupSecurityAccess` interface.

1.4.2.7.2 Functional Security

Users with similar functions are typically organized into user groups (instances of `GroupAccount`). The rights assigned to the group reflect the functional security rights of the users in the group. You

cannot modify functional rights using the TeamConnect API, only through the user interface. If you want to know if a user has a particular functional right, you can use the following interfaces:

- `SecurityService`—Provides methods that can check a user's rights within TeamConnect.
- `OperationType`—Provides enumerations that include all the rights a user can have.

Some of the `SecurityService` methods use the `OperationType` class. For example, the following code checks if a user can create an account:

```
boolean hasEntityOperationRights =
platform.getSecurityService().isEntityOperationEnabled(user,
OperationType.CREATE, "ACCT");
```

In the previous code snippet, the `securityService.isEntityOperationEnabled()` method checks if the given user, called **anyUser**, has the necessary rights to create an account object. If the user has the rights, this code returns `true`.

In addition to a method for checking rights at the object level, `SecurityService` also includes methods for checking whether a user has rights to categories, tools, and user invoked rules:

- `isCategoryOperationEnabled()` checks rights for categories.
- `isToolAccessEnabled()` checks rights for system or custom tools.
- `isUserInvokedAccessEnabled()` checks rights for user invoked rules.
- `isEntityOperationEnabled()` checks rights for objects.

1.4.2.8 System and User Settings

You can use the API to retrieve and update system and user settings. System settings are settings an administrator updates from the **Admin Settings**. User settings are settings that a user updates on their **Preferences** page. When working with these settings in the API, you use `SettingsService` with the `SettingKey` enums.

SettingsService Interface

`SettingsService` provides methods to retrieve and update system and user settings. This interface includes the following types of methods:

- *Methods that retrieve the value for a setting*—Most `SettingsService` methods retrieve a value for a specific setting. For example, the following method returns the filename of the custom logo image.

  ```
  settingsService.getSystemCustomLogoImage();
  ```

  For settings that do not have a specific get method, you can use the `getSystemSetting()` or `getUserSetting()` methods. You can also use these methods for settings that do have a get method. For example, the following code returns the filename of the custom logo image, just as the `getSystemCustomLogoImage()` method does.

  ```
  settingsService.getSystemSetting(SettingsKey.CUSTOM_LOGO_IMAGE);
  ```

Note: *The getSystemSetting() and getUserSetting() methods always return String values. Convert each value to another data type, such as an Integer or Boolean, if necessary. Get methods for specific settings, such as getSystemMaxUploadSize(), return the appropriate data type for the TeamConnect field.*

- *Methods that set the value of a setting*—You use the `setSystemSetting()` and the `setUserSetting()` methods to set the values for system and user settings. `SettingsService` does not provide methods for setting specific settings.

Note: *Use the following code to access SettingsService:* `SettingsService settingsService = platform.getSettingsService();`

Two Types of Setting Parameters

Two methods of the same name exist for each of the following previously mentioned methods: `getSystemSetting()`, `getUserSetting()`, `setSystemSetting()`, and `setUserSetting()`. These two methods are different because they each use two different types of parameters:

- The enums in the `SettingKey` class. This class includes enums for all user and system settings.
- The key of a custom setting. You can only add a custom setting through the TeamConnect API, not from the TeamConnect **Setup**.

For an example of the different types of parameters, the next two code samples use the two different `setSystemSetting()` methods.

In the following sample, `setSystemSetting()` uses a `SettingKey` enum as a parameter. Using the SEARCH_MAX_TIME setting key, this code updates the **Maximum Search time (seconds)** field in the **Admin Settings** to a value of 25.

```
settingsService.setSystemSetting(SettingsKey.SEARCH_MAX_TIME, "25");
```

The following sample includes a custom field as a parameter for `setSystemSetting()`. This code updates a custom field in the **Admin Settings**, specified by the **Custom System Setting** parameter, to a value of 25.

```
settingsService.setSystemSetting("Custom System Setting", "25");
```

1.4.2.8.1 Adding Custom Settings

You can add user and system custom settings to TeamConnect using the API. When you add a custom setting, you can enter custom code so that the setting works throughout TeamConnect.

To add custom settings, use the `setSystemSetting()` and `setUserSetting()` methods that receive the custom setting name parameter. Even though custom setting values save as String values, you can convert the setting to another data type as part of the code.

For example, if you want to create a new user setting with the **New Custom Setting** key, use the following code:

```
// Adds a new custom setting to the system.
// Making the same call after the setting is initially added updates the value
like it does for non-custom settings.
platform.getSettingsService().setSystemSetting("New Custom Setting", "10");
// Setting values save as a String, so you can convert the String to an Integer
or another data type.
// You can enter custom code to check the value of the newly added setting and
impact parts of TeamConnect.
if(Integer.valueOf(platform.getSettingsService().getSystemSetting("newCustomSett
ing")) > 5) {
    // Custom code here
}
```

Note: *If* ***New Custom Setting*** *already exists when you enter the previous code, the setting updates with the new value.*

1.4.2.9 Internationalization

TeamConnect provides ways for you to customize system and custom data for multiple languages and regions. In addition to changing the language in the user interface, you can also change the currency and the time and date display for a specified region.

`InternationalizationService` is the main interface for working with different locales and currencies. None of the methods for this interface change anything in your version of TeamConnect. However, you can use the methods to retrieve information based on specified locales. If an `InternationalizationService` method does not require a locale, the code uses the current user's locale.

`InternationalizationService` includes methods for the following types of actions:

- *Converting currency*—You can use the `convertCurrency()` method to convert one type of currency to the system currency. The following code sample converts 100 British Pounds to the system currency.

```
BigDecimal value = new BigDecimal(100);
BigDecimal valueInOtherCurrency =
platform.getInternationalizationService().convertCurrency(value, "GBP");
```

- *Retrieving the currency based on a code*—You can use the `getCurrency()` method to retrieve a currency. The following code sample returns the currency object for GBP, which includes the currency's code, symbol, and exchange rate.

```
Currency currency = internationalizationService.getCurrency("GBP");
```

- *Retrieving values for a system or user setting*—For example, you can use the `getCurrentUserLocale()` method to return the locale of the current user, as shown in the following code sample.

```
Locale currencyUserLocale =
internationalizationService.getCurrentUserLocale();
```

- *Returning a localized version of a string or number*—For example, you can use the `localizeNumber()` method to determine the format of a number based on the locale. The following code sample returns 5.555,55 for the French locale.

```
String localizedNumberString = internationalizationService.localizeNumber(new
BigDecimal(5555.55), Locale.FRANCE);
```

  You can also use the `localize()` method with a custom message to format the date.

- *Sorting based on a locale*—Based on the current user's locale, you can use the `sortCategories()` method or the `sortLookupItems()` method to sort items with the same display order alphabetically, as shown in the following code sample.

```
List<LookupItem> sortedLocalizedLookupItems =
internationalizationService.sortCategories(unsortedLookupItems);
```

Note: *To update the display order, use the CategoryDefinition interface for categories and the LookupItem interface for lookup items.*

Note: *Use the following code to access InternationalizationService:* `InternationalizationService internationalizationService = platform.getInternationalizationService();`

1.4.2.10 Searching for Records

In your API code, you may need to perform actions on records that meet specified criteria, particularly as part of a rule specification. For example, before integrating with a billing system, a rule may search the database for all task records added to a matter within a period of time.

Search in the API using `ResourceService.search()`, which provides similar functionality to search views in the TeamConnect user interface.

Because searching may drastically affect performance, verify that there is no other way to get the desired list of records. If you must conduct a search, especially against a large number of records, narrow your search criteria and consider providing a time-out.

Before using the `search()` methods, consider using available predefined methods to get related records. For example, the `User` interface has the `getContact()` method to obtain the contact of a user. The `Project` interface has methods for obtaining a list of child or involved records.

The following table provides examples of records that custom code is likely to includes as part of a search and whether or not an alternative exists.

To Search or Not to Search

If you are looking for	Use search?		Use this instead
	Yes	No	

All contacts.	x		
Any documents (but not document folders).	x		
Any history records.	x		
Any of these system object records related records to a Project record: • Accounts • Appointments • Expenses • Tasks	x		
Child or embedded records of a `Project` record.		x	`Project.getEmbeddedProjects` `ProjectService.getChildProjects`
Child records of an `Account` record.	x		
Current user.		x	`UtilityService.getCurrentUser()`
Involved record associated with an account.	x		
Old object.		x	`ResourceService.read()`
Parent record of a related `Account` record.		x	`Account.getParentAccount()`
Parent record of a related `Project` record.		x	`Project.getParentProject()`
Project of a related system object record.		x	*`SystemObject`*`Name.getProject()` For example, `Task.getProject()`
Project record of an `Involved` record.		x	`Involved.getProject()`
Record to which a `History` record is related.		x	`History.getParentObject()`

Sub-objects.		x	Use the appropriate methods defined in the owner interface. For details, see Sub-Objects.

1.4.2.10.1 Creating an API Search View

When you are creating code for searching, you must include the following information in your search code:

- One of three `ResourceService.search()` methods to execute the search.
  - A method that uses the search view key to return the results of a search view.
  - A method that uses filter criteria to return search results.
  - A method that uses filter criteria and search parameters to return search results.
- Methods part of the search classes to enter search criteria and parameters.
- The resource service class and concrete objects to specify the objects you are searching. For example, if you are searching for accounts, access `AccountService` so that you can use `Account` methods as part of the search.

When you include all this information, you can execute the search.

1.4.2.10.1.1 Search Criteria

Creating search criteria is the same as entering **Filter Criteria** for a custom search. To create filter criteria, use one of the two `search()` methods that accept `SearchCriteria`. Depending on your filter criteria, you can construct `SearchCriteria` in one of three ways:

- `SearchCriteria()`—Searching for one type of record.
- `SearchCriteria(FieldCriterion fieldCriterion)`—Searching with one filter criteria.
- `SearchCriteria(SearchExpression expr1, SearchExpression expr2, SearchExpression... expressions)`—Searching with multiple filter criteria.

Searching for One Type of Record

To return all records of a specific type, use an empty `SearchCriteria()`. For example, if you want to return all records for one project type, pass in the project's unique code and `SearchCriteria()`, as shown in the following sample.

```
// Using an empty search criteria to return all Disputes
List<Project> allDisputes = platform.getProjectService().search("DISP", new
SearchCriteria());
```

Searching with One Filter Criteria

To search with one filter criteria, use `SearchCriteria(FieldCriterion fieldCriterion)`. Use `FieldCriterion` to enter the fields you are searching on and its subclasses when you know the

data types of those fields. Along with the field, you include the value of the field that determines the search results.

For example, if you want to search for all appointments in Room 101, you can use the code in the following sample.

```
// Retrieve the appointment service from the platform provided by the API.
AppointmentService appointmentService = platform.getAppointmentService();

// Find all appointments located in Room 101
StringCriterion locationCriterion = new
StringCriterion(Appointment.LOCATION).equalTo("Room 101");
SearchCriteria criteria = new SearchCriteria(locationCriterion);

// Execute search
List<Appointment> results = appointmentService.search(criteria);
```

The previous sample uses the following code to build search criteria:

- `Appointment.LOCATION` for searching on a system field.
- `equalTo("Room 101")` for setting the operator.

Searching with Multiple Filter Criteria

To search with multiple filter criteria, use `SearchCriteria(SearchExpression expr1, SearchExpression expr2, SearchExpression... expressions)`. `SearchExpression` allows you to search during the following scenarios.

All or Any Criteria as Part of the Results

You may want to search with more than one filter criteria and you want all or any of the criteria to be part of the results. For all the criteria, use the `and()` method. For any of the criteria, use the `or()` method.

The following code sample provides an example of using the `or()` method to join date criteria and combining the date criteria with the `and()` method.

```
// Retrieve the appointment service from the platform provided by the API.
AppointmentService appointmentService = platform.getAppointmentService();

// Find all appointments located in Room 101
SearchCriteria criteria = new SearchCriteria(new
StringCriterion(Appointment.LOCATION).equalTo("Room 101"));

// Find appointments starting within four days or ending in the next two days
RelativeDateCriterion startOnCriterion = new
RelativeDateCriterion(Appointment.START_ON).next(4);
```

```
RelativeDateCriterion endOnCriterion = new
RelativeDateCriterion(Appointment.END_ON).next(2);

// Join the two date criteria, specifying 'or' logic
SearchCriteria dateCriteria = startOnCriterion.or(endOnCriterion);

// Join the date criteria and location criteria
criteria.and(dateCriteria);

// Execute search and return results
List<Appointment> results = appointmentService.search(criteria);
```

The previous sample includes the following code:

- The first search expression uses one filter criteria to specify that the results return all appointments in Room 101.

```
SearchCriteria criteria = new SearchCriteria(new
StringCriterion(Appointment.LOCATION).equalTo("Room 101"));
```

- The second search expression specifies relative date criteria using the `next()` method and combines them with the `or()` method to specify whether the search returns the applicable start date or end date.

```
RelativeDateCriterion startOnCriterion = new
RelativeDateCriterion(Appointment.START_ON).next(4);
RelativeDateCriterion endOnCriterion = new
RelativeDateCriterion(Appointment.END_ON).next(2);


SearchCriteria dateCriteria =
startOnCriterion.or(endOnCriterion);
```

- The sample combines both of the previous expressions.

```
criteria.and(dateCriteria);
```

- The code executes.

```
List<Appointment> results = appointmentService.search(criteria);
```

Complex Criteria to Determine Results

You may want to specify whether some and which criteria should be, should possibly be, or should not be part of the results. You can use a different method for each search criteria:

- You can specify that the search must include two search conditions with `and()`.
- You can specify that either search condition should appear in the search results using `or()`.

- You can specify conditions for results that you do not want to include using `not()`.

The following code sample provides an example of the previous scenario.

```
// Get service
AppointmentService appointmentService = platform.getAppointmentService();

// Group One: In Room 101 AND All Day
StringCriterion locationCriterion = new
StringCriterion(Appointment.LOCATION).equalTo("Room 101");
BooleanCriterion allDayCriterion = new
BooleanCriterion(Appointment.ALL_DAY).isTrue();
SearchCriteria andGroup = locationCriterion.and(allDayCriterion);

// Group Two: Started within the last 5 days OR ends in the next 3 days
RelativeDateCriterion startOnCriterion = new
RelativeDateCriterion(Appointment.START_ON).withinLast(5);
RelativeDateCriterion endOnCriterion = new
RelativeDateCriterion(Appointment.END_ON).next(3);
SearchCriteria orGroup = startOnCriterion.or(endOnCriterion);

// Composite Group: NOT in both group one and group two
SearchCriteria notGroup = new SearchCriteria(andGroup, orGroup).not();

// Execute search and return results
return appointmentService.search(notGroup);
```

The previous sample includes the following code:

- The first search expression specifies all day-long appointments in Room 101.

```
StringCriterion locationCriterion = new
StringCriterion(Appointment.LOCATION).equalTo("Room 101");
BooleanCriterion allDayCriterion = new
BooleanCriterion(Appointment.ALL_DAY).isTrue();
SearchCriteria andGroup = locationCriterion.and(allDayCriterion);
```

- The second search expression specifies all appointments that started in the last 5 days or end in the next 3 days.

```
RelativeDateCriterion startOnCriterion = new
RelativeDateCriterion(Appointment.START_ON).withinLast(5);
RelativeDateCriterion endOnCriterion = new
RelativeDateCriterion(Appointment.END_ON).next(3);
SearchCriteria orGroup = startOnCriterion.or(endOnCriterion);
```

- The third search expression specifies all appointments not part of the other two expressions.

```
SearchCriteria notGroup = new SearchCriteria(andGroup, orGroup).not();
```

- The sample searches for all results that the third expression returns as part of its execution.

```
appointmentService.search(notGroup);
```

Note: *This code uses the* `and(SearchExpression expression)` *method instead of the* `and()` *method because it combines multiple filter criteria with additional search expressions. If the AND or OR applies to all filter criteria, use the* `and()` *or* `or()` *methods.*

When you have one field that is directly part of a record, you can construct a criterion with the `SearchableField`.

Note: *If the record does not have direct access to the field but can access it through a field in the record, you must construct a criterion with a FieldPath.*

The type of criterion depends on the type of field. The API provides the following criterion types with `SearchableField` constructors:

- `BooleanCriterion`—For Boolean fields.
- `CategoryCriterion`—For categories.
- `DateCriterion`—For date fields.
- `DecimalCriterion`—For BigDecimal fields.
- `EnumerationCriterion`—For enum-based fields.
- `FieldCriterion`—For all fields.
- `IntegerCriterion`—For integer fields.
- `LookupItemCriterion`—For lookup table fields.
- `ObjectCriterion`—For account, contact, project, and user fields.
- `RelativeDateCriterion`—For integers before and after a certain date.
- `StringCriterion`—For string fields.

The `SearchableField` you use depends on whether the field is a system or custom field.

System Field Searching

When you want to search for a system field, use the static field in the model class for the object. You can find a list of static fields for each class in the javadocs: **~\utilities\javadocs\api\index.html**

For example, if you are searching on the **Location** field in appointments, the `LOCATION` static field is part of the `Appointment` class. As a result, the `SearchableField` is `Appointment.LOCATION`. Before you add the operator, the search field of the criterion looks like the following code.

```
StringCriterion locationCriterion = new StringCriterion(Appointment.LOCATION)
```

Custom Field Searching

When you want to search for custom fields, you can use the `CustomField` class to retrieve information about fields for a search.

Note: *When searching on system fields, use the static system fields in the class for that object. For example, to search with the* ***Location*** *field in appointments, use Appointment.LOCATION.*

Use the `CustomField` class when you have to include custom fields in search criteria. You can use this interface with custom fields of all data types. However, if you are working with projects, Involved parties, or lookup table custom fields, you have the following additional classes:

- `ProjectCustomField`
- `LookupCustomField`
- `InvolvedCustomField`

The following code sample provides an example of how to search using a custom field as the search criteria.

```
// Get instances of the services we'll need
ProjectService projectService = platform.getProjectService();
CategoryService categoryService = platform.getCategoryService();


// Retrieve the custom field
CustomField<String> textField =
categoryService.getCustomField("fullTreePositionofCategory", "customFieldName");


// Find records with the given custom field set to "Test value"
StringCriterion stringCriterion = new StringCriterion(textField).equalTo("Test
value");
SearchCriteria searchCriteria = new SearchCriteria(stringCriterion);


// Execute search and return results
return projectService.search("DISP", searchCriteria");
```

Field paths contain more than one `SearchableField`. Use `FieldPath` to construct a field when the record does not have direct access to the field you want to search.

For example, if you want to search with the username of an appointment attendee, you cannot access the user directly from the appointment. The `Appointment` class contains the static ATTENDEES field. Once you have access to the attendees, you can access the static USER field with the `AppointmentAttendee` class. After you have access to the user, you can access the static USERNAME field with the `User` class. As a result, the `FieldPath` is `(Appointment.ATTENDEES).add(AppointmentAttendee.USER).add(User.USERNAME);`

Before you add the operator, the field path looks like the following code.

```
FieldPath fieldPath = new
FieldPath(Appointment.ATTENDEES).add(AppointmentAttendee.USER).add(User.USERNAME
);
```

The following code includes an example of a search with two criteria that use field paths.

```
AppointmentService appointmentService = platform.getAppointmentService();

// Create a path to the Appointment's Project's Created On
FieldPath projectCreatedOn = new
FieldPath(Appointment.PROJECT).add(Project.CREATED_ON);

// Create a path to the Appointment's Attendee's Attendance Type
FieldPath attendeeAttendanceType = new
FieldPath(Appointment.ATTENDEES).add(AppointmentAttendee.ATTENDANCE_TYPE);

// Search for appointments related to projects created in the last thirty days
RelativeDateCriterion relativeDateCriterion = new
RelativeDateCriterion(projectCreatedOn).withinLast(30);

// Search for appointments that have attendees that will not attend
EnumerationCriterion<AttendanceType> enumerationCriterion = new
EnumerationCriterion<AttendanceType>(attendeeAttendanceType).equalTo(AttendanceT
ype.WILL_NOT_ATTEND);

// Criterions constructed from field paths behave exactly like criterions
constructed from fields, and can be combined just the same
SearchCriteria criteria = new SearchCriteria(relativeDateCriterion,
enumerationCriterion);

// Execute search
List<Appointment> appointments = appointmentService.search(criteria);
```

You can also use field paths with custom field searching, as shown in the following sample.

```
ProjectService projectService = platform.getProjectService();
CategoryService categoryService = platform.getCategoryService();

// Field Paths can also point to custom fields
CustomField<String> customField = categoryService.getCustomField("CONT", "custom
field");
FieldPath contactCustomField = new FieldPath(Project.CONTACT).add(customField);

// Search for projects based on contact custom field
```

```
StringCriterion stringCriterion = new
StringCriterion(contactCustomField).equalTo("value");
SearchCriteria criteria = new SearchCriteria(stringCriterion);

// Execute search
List<Project> projects = projectService.search("DISP", criteria);
```

Setting operators in the API are similar to setting operators in TeamConnect custom searches. To search with operators, use the `FieldCriterion` class and all the classes that extend it. `FieldCriterion` includes methods for operators that you can use on all fields.

Because each data type has different operators, each data type class has its own set of methods for using operators. These methods are part of the following classes:

- `BooleanCriterion`—Operators for Boolean fields.
- `CategoryCriterion`—Operators for categories.
- `DateCriterion`—Operators for date fields.
- `DecimalCriterion`—Operators for BigDecimal fields.
- `EnumerationCriterion`—Operators for enum-based fields.
- `IntegerCriterion`—Operators for integer fields.
- `LookupItemCriterion`—Operators for lookup table fields.
- `ObjectCriterion`—Operators for account, contact, project, and user fields.
- `RelativeDateCriterion`—Operators for integers before and after a certain date.
- `StringCriterion`—Operators for string fields.

In the case of the example for system field searching without the operator, the entire criterion looks like the following code with the operator:

```
StringCriterion locationCriterion = new
StringCriterion(Appointment.LOCATION).equalTo("Room 101");
```

1.4.2.10.1.2 Search Parameters

Use search parameters to control the search, such as the page size and the sort order of the search results. To control your search, use the search() method that accepts the `SearchParameters` class.

As you create the code for searching, use the search parameter constructors. The constructors include a combination of the following search parameters:

- `beginIndex`—The first page of results the search returns.
- `limit`—The number of search results to return on a page.
- `useUnsavedChanges`—An indication of whether the search should run against changes not yet committed to the database.

- `sortFields`—The fields for sorting the search results.

If a constructor does not include one of these parameters, the search uses the default for the parameter. Retrieve or set the default settings using the `SearchParameters` get and set methods (`getBeginIndex()`, `setLimit()`, etc.).

The following list provides examples of some SearchParameters constructors. For a full list of constructors, refer to the `SearchParameters` class in the javadocs.

- `SearchParameters(SortField... sortFields)`

  You can specify how you want the search results to sort. This constructor creates a parameter with one or more sort fields, the default limit, and the default begin index. The following code sample sorts the results by start date in ascending order.

```
// Sort results by start date
SearchParameters parameters = new SearchParameters(new
SortField(Appointment.START_ON));
```

- `SearchParameters(java.lang.Integer limit)`

  You can limit the number of results in a search. This constructor creates a parameter with the number of search results, the default being index, and no sort fields. The following code sample limits the number of results to 10 records.

```
// Limit results to 10 records
SearchParameters parameters = new SearchParameters(10);
```

- `SearchParameters(boolean useUnsavedChanges)`

  You can specify whether a search should return results from the database or changes not yet commented to the database. This constructor creates a parameter that specifies which to search, the default limit and begin index, and no sort fields. The following code sample specifies that the search results will include changes not yet saved to the database.

```
// Includes unsaved changes in search results
SearchParameters parameters = new SearchParameters(true);
```

1.4.2.10.2 Executing a Search

When you have all the information you need to return search results, whether it is the search view key or the search criteria and parameters, you can execute the search.

The following code sample executes a search with search criteria and parameters. This sample returns search results for appointments in Room 101 sorted by start date.

```
public class AppointmentSample extends List<Appointment> {

@Override
    public void action(Appointment appointment) {
    }
```

```
    public void searchAppointment() {

        // Retrieve the appointment service from the platform provided by the
        API.
        AppointmentService appointmentService = platform.getAppointmentService();

        // Find all appointments located in Room 101
        StringCriterion locationCriterion = new StringCriterion(new
        FieldPath(Appointment.LOCATION)).equalTo("Room 101");
        SearchCriteria criteria = new SearchCriteria(locationCriterion);

        // Sort results by start date
        SearchParameters parameters = new SearchParameters(new SortField(new
        FieldPath(Appointment.START_ON)));

        // Execute search
        List<Appointment> results = appointmentService.search(criteria,
        parameters);

    }
}
```

1.4.2.11 Logging

TeamConnect's logging functionality allows the system to generate logging messages for information or troubleshooting. As a TeamConnect developer, you can specify messages for log levels during customization, design, and production.

For example, when you are writing and testing code, include **Debug** messages so that you can see what occurs when your code executes. When code is in production, messages with a higher level of severity, such as **Warn** or **Error**, indicate less frequent but more critical problems in your code.

Logs capture messages based on the logger levels that system administrators or solution developers define in TeamConnect. TeamConnect only logs messages for the level defined and the levels of higher severity. As a result, if you include a message in your code for a lower level, TeamConnect does not log the message to the logger. For example, loggers with a **Warn** level include messages with levels of **Warn**, **Error**, and **Fatal**. If your code includes a message for the **Info** logger level, the message does not log.

Loggers for API Code

TeamConnect has several predefined loggers. Depending on the type of code you are writing, TeamConnect logs messages to the corresponding logger:

- Code for an automated qualifier or action logs messages to the **Rule** logger.
- Code for a custom screen logs messages to the **Custom Blocks** logger.
- Code for a custom tool logs messages to the **Tools** logger.

Logging Messages from the API

Complete the following steps for your custom code to generate log messages:

1. On the **Logging** page of the **Admin Settings**, define the logging level for the type of logger associated with the code.
2. Specify logging messages in your code using the `CustomItem` or `Platform` logging methods.
3. Add code that checks the logging levels in TeamConnect before running logging methods.

1.4.2.11.1 Specifying Logging Messages

The following classes include methods for logging:

- `CustomItem`—Includes the logging methods for code that extends `CustomItem`, which means automated qualifiers, automated actions, custom screens, and custom tools.
- `Platform`—Includes the logging methods for code that does not extend `CustomItem`.

Both classes include the same logging methods at the following logger levels:

- **Debug**
  - `logDebug(String message)`
  - `logDebug(String message, Throwable t)`
- **Info**
  - `logInfo(String message)`
  - `logInfo(String message, Throwable t)`
- **Warn**
  - `logWarn(String message)`
  - `logWarn(String message, Throwable t)`
- **Error**
  - `logError(String message)`
  - `logError(String message, Throwable t)`

The following code includes examples of methods that log messages:

```
String username = "username";
Date time = new Date();

// Log some debugging information
logDebug("rule executed by " + username + " at " + time);
// Also possible via Platform
platform.logDebug("rule executed by " + username + " at " + time);

// Log an informational message
```

```
logInfo("rule executed by " + username + " at " + time);
// Also possible via Platform
platform.logInfo("rule executed by " + username + " at " + time);

// Log a warning
logWarn("Missing some information");
// Also possible via Platform
platform.logWarn("Missing some information");

// Log an error
logError("This is a problem");
// Also possible via Platform
platform.logError("This is a problem");
```

1.4.2.11.2 Checking Logger Levels

Running code with logging methods can have an impact on performance. You can prevent code from running unnecessarily by first checking whether the specified logger level is enabled on the **Logging** page of **Admin Settings**.

The API provides the following methods for checking whether the **Debug** or **Info** logger levels are enabled:

- `isDebug()` to check for the **Debug** logger level.
- `isInfo()` to check for the **Info** logger level.

Note: *These levels have methods for checking because levels of a higher severity are not intended for code in production.*

The following code for logging **Debug** and **Info** log messages checks whether the logger levels are turned on before running the log methods.

```
// Log some debugging information
logDebug("rule executed");
// Check the log level before executing complex debugging statements to improve
performance
if(isDebug()) {
logDebug("rule executed by " + username + " at " + time);
}

// Log an informational message
logInfo("rule executed");
// Check the log level before executing complex informational statements to
improve performance
if(isInfo()) {
logInfo("rule executed by " + username + " at " + time);
```

```
}
```

1.4.3 Java Samples

This reference section provides examples of automated qualifiers, automated actions, and searching samples written in Java. If desired, you can copy these samples and use them as models or starting points for your own custom code.

The concepts that are demonstrated in these samples are introduced in The TeamConnect API, Automated Qualifiers and Actions, and Searching for Records.

Sample Values

Each complete sample includes a description and a table listing all of the example values in the sample. The values in each sample are intended to demonstrate the kind of values needed, such as unique codes of object definitions, full tree positions of categories, and custom field names.

System User In These Samples

Many of the following code samples switch to the system user to enable the code to be executed regardless of the current user's rights.

However, you must determine whether switching to the system user is necessary in your own code based on the requirements for your custom code and your custom TeamConnect design. For more details about the system user, see Running Code as the System User.

1.4.3.1 Partial Samples to Demonstrate Concepts

The following code samples are incomplete. They demonstrate concepts as noted in the heading and description. These concepts are presented in the context of rule qualifiers, but you can apply them in other types of custom code. You can use these qualifiers with any rule type.

You can find the following partial code samples in this section:

- Checking Lists of Added Categories in Records
- Looping Through Related Records of Projects
- Getting Child Projects of Specific Custom Object

Checking Lists of Added Categories in Records

The following rule qualifier code sample in Java shows how you can check whether a specific category is one of the multiple categories added to a record.

Java Sample: Checking Lists of Added Categories

```
import java.util.List;
import com.mitratech.teamconnect.enterprise.api.model.Category;
import com.mitratech.teamconnect.enterprise.api.model.Project;
public class CheckingValuesInRelatedRecords3 extends CustomCondition<Project> {
```

```
    @Override
    public boolean condition(Project record) {
        // Getting list of categories added to the project
        List<Category> categories = record.getCategories();
        for (Category category : categories) {
            // Looking for some specific category
            if (category.getTreePosition().equals("FULL_TREE_POSI_TION")) {
                return false;
            }
        }
        return true;
    }
}
```

Looping Through Related Records of Projects

Related records of projects, including involved records and child projects (custom objects), have specific methods for getting them. These methods are described in Related Records.

You retrieve a list of Involved parties or all child projects:

Java Sample: Looping Through Related Records of Projects

```
public void action(Project project) {
    // Get list of child objects for project for a specific custom object
    definition
    List<Project> children =
    platform.getProjectService().getChildProjects(project,     "uniqueCode");

    // Loop through list of child objects
    for (Project child : children) {
        // Do something with the child here
    }

    // Get list of embedded entities for the same specific custom object
    definition
    List<EmbeddedEntity> embeddedEntities=
    project.getEmbeddedEntities("uniqueCode");

    // Loop through list of embedded entities
    for (EmbeddedEntity entity : embeddedEntities) {
        // Do something with the entity here
    }
}
```

Getting Child Projects of Specific Custom Objects

The following code sample obtains a list of child or embedded objects of a specific custom object definition using its unique code. The method for getting the list takes the unique code of that object definition as an argument:

Java Sample: Getting Child Projects of Specific Custom Objects

```
public void action(Project project) {
    // Get list of child objects for project for a specific custom object
    definition
    List<Project> children =
    platform.getProjectService().getChildProjects(project, "uniqueCode");
    // Loop through list of child objects
    for (Project child : children) {
        // Do something with the child here
    }
    // Get list of embedded entities for the same specific custom object
    definition
    List<EmbeddedEntity> embeddedEntities=
    project.getEmbeddedEntities("uniqueCode");
    // Loop through list of embedded entities
    for (EmbeddedEntity entity : embeddedEntities) {
        // Do something with the entity here
    }
}
```

1.4.3.2 Automated Qualifiers

You can use the following examples of automated qualifiers written in Java with any rule type.

Depending on whether you need the old object or the current object, different methods are necessary for getting values of an object. In these examples, the values in the current object are used unless otherwise stated.

You can find the following automated qualifier samples in this section:

- Checking System Field Values
- Comparing Custom Field Values
- Checking Whether System Field Values Were Changed
- Checking Whether Custom Field Values Were Changed
- Checking Roles in Related Records
- Checking Default Category in Related Records
- Checking Values in Vendor's Contact Record
- Checking Whether Custom Field Has a Value

Checking System Field Values

Both the total amount and activity item are system fields in task records (instances of `Task`). Therefore, the interface `Task` has specific methods to get their values.

The activity items in task records are system lookup table items. For more details, see Lookup Tables and List Fields.

Business Rule	No tasks that are over $10,000 with the activity item Printing can be posted without approval.
Qualifier	Checks whether: • The total task amount is over $10,000. • The task's activity item is Printing.
Sample Values	**Object Definition:** TASK (system object Task) **Task Activity Item:** ACTI_PRNT (Printing) **Categories:** None **Custom Fields:** None

Java Qualifier: Checking System Field Values

```
public class VerifyingSystemFieldValues extends CustomCondition<Task> {
    @Override
    public boolean condition(Task task) {
        // Check whether the activity item of the task is Printing and the total
        amount of the task is greater than or equal to 10,000
        return task.getActivityItem().equals("ACTI_PRNT") &&
        task.getTotalAmount().doubleValue() >= 10000;
    }
}
```

Comparing Custom Field Values

Automated qualifiers are frequently used to compare field values. You can compare the values of any fields as long as the values are of the same data type. This example demonstrates how to compare the values from two custom fields of type Date.

Business Rule	Served company records do not save if the values in the **Service Date** and **Received Date** custom fields are not the same.

Qualifier	Checks whether the date values are in two custom fields of the same type.
Sample Values	**Object Definition:** LITI (custom object Litigation) **Categories:** LITI (root category of Litigation) **Custom Fields:** ServiceDate (Date) ReceivedDate (Date)

Java Qualifier: Comparing Custom Field Values

```
public class ComparingTwoCustomFieldValues extends CustomCondition<Project> {
    @Override
    public boolean condition(Project project) {
        // Getting the value of the service date field
        Date serviceDate = project.getDateFieldValue("LITI", "serviceDate");

        // Getting the value of the received date field
        Date receivedDate = project.getDateFieldValue("LITI", "receivedDate");

        // Return true if they are not equal
        return !serviceDate.equals(receivedDate);
    }
}
```

Checking Whether System Field Values Changed

Automated qualifiers in business rules often compare the values of a record that displayed when the record opened with the values entered by the currently logged-in user before saving the record.

The following sample checks whether the contact-centric field of a record has changed.

Business Rule	Main policy holder cannot change in policy records without approval.
Qualifier	Checks the value on the screen in the **Policy Holder** contact-centric field against the value in the database.
Sample Values	**Object Definition:** Policy or any custom object definition **Categories:**

	None **Custom Fields:** None

Java Qualifier: Checking Whether System Field Values Changed

```
public class CheckingWhetherRecordValuesWereChanged extends
CustomCondition<Project> {
    @Override
    public boolean condition(Project project) {
        // Getting the object as it currently exists in the database - that is,
        the 'old' object
        Project initialProject =
        platform.getProjectService().readLastSaved(project);

        // Return true if the contact associated with the project has been
        changed
        return project.getContact() != null &&
        initialProject.getContact().getPrimaryKey() !=
        project.getContact().getPrimaryKey();
    }
}
```

Checking Whether Custom Field Values Changed

This sample demonstrates how to check whether the value in a custom field changed. Notice that the method of getting the database value and the value entered during the current session is done in the same as in the previous sample.

Business Rule	Main policy holder cannot change in policy records without approval.
Qualifier	Checks the value on the screen in the **Policy Holder** custom field against the value in the database.
Sample Values	**Object Definition:** PLCY (custom object Policy) **Categories:** PLCY (root category of Policy) **Custom Fields:** PolicyHolder (Involved)

Java Qualifier: Checking Whether Custom Field Values Changed

```
public class CheckingWhetherCustomFieldValuesWereChanged extends
```

```
CustomCondition<Project> {
    @Override
    public boolean condition(Project project) {
        // Getting the object as it currently exists in the database - that is,
        the 'old' object
        Project initialProject =
        platform.getProjectService().readLastSaved(project);


        // Getting new policy holder
        Involved currentPolicyHolder = project.getInvolvedFieldValue("PLCY",
        "PolicyHolder");


        // Getting old policy holder
        Involved initialPolicyHolder =
        initialProject.getInvolvedFieldValue("PLCY", "PolicyHolder");


        // Return true if the policy holder has been changed
        return currentPolicyHolder.getPrimaryKey() !=
        initialPolicyHolder.getPrimaryKey();
    }
}
```

Checking Roles in Related Records

You can use automated qualifiers to check the values in related records of the current object.

Business Rule	No claim records can update if they do not have an involved record with the role Claimant.
Qualifier	Checks if an involved record has the specified role.
Sample Values	**Object Definition:** Policy or any custom object definition **Related Object of Current Object:** CLIN (Involved) **Category in Related Object:** CLIN_CLMT (Involved role Claimant) **Custom Fields:** None

In the following sample, after getting the list of Involved records related to the current project object, the rule checks the list of categories (involved roles) added to each involved record. If a role does not exists, the qualifier returns true.

Java Qualifier: Checking Roles in Related Involved Records

```
public class CheckingValuesInRelatedRecords extends CustomCondition<Project> {
    @Override
    public boolean condition(Project project) {
        List<Involved> involvedList =
        platform.getProjectService().getInvolved(project);
        for (Involved involved : involvedList) {
            List<Category> roles = involved.getCategories();
            for (Category role : roles) {
                if (role.getTreePosition().equals("CLIN_CLMT")) {
                    // Return false if the project has any involved with the role
                    of Claimant
                    return false;
                }
            }
        }
        // Return true if the project has no involved with the role of Claimant
        return true;
    }
}
```

Checking Default Category in Related Records

The following sample gets the list of Involved records related to the current object and checks the default category (role) of each Involved record. If none of the Involved records have a particular category, the qualifier returns true.

Business Rule	Claim records can only update if they have an Involved record with the role Claimant.
Qualifier	Checks if there is an Involved record with the specified role as its default role (category).
Sample Values	**Object Definition:** Claim or any custom object definition **Related Object of Current Object:** CLIN (Involved) **Category in Related Object:** CLIN_CLMT (Involved role Claimant) **Custom Fields:** None

Java Qualifier: Checking the Default Role in Related Involved Records

```
package com.mitratech.teamconnect.enterprise.api.model.rule;
import java.util.List;
import com.mitratech.teamconnect.enterprise.api.model.Involved;
import com.mitratech.teamconnect.enterprise.api.model.Project;
import com.mitratech.teamconnect.enterprise.api.service.ProjectService;
public class CheckingValuesInRelatedRecords2 extends CustomCondition<Project> {
    @Override
    public boolean condition(Project project) {
        // Getting the project service
        ProjectService projectService = platform.getProjectService();
        // Getting the list of involved records related to the project
        List<Involved> involvedList = projectService.getInvolved(project);
        for (Involved involved : involvedList) {
            // If the default category is Claimant, return false
            if
            (involved.getPrimaryCategory().getTreePosition().equals("CLIN_CLMT"))
            {
                return false;
            }
        }
        return true;
    }
}
```

Checking Values in Vendor's Contact Record

The following qualifier checks whether the vendor of an invoice has a specific checkbox selected in the contact record. The qualifier then checks whether the number of line items in the invoice is more than the number that were in the previously saved version of the invoice.

Business Rule	If the contact selected in the **Vendor** field of an invoice has the **Volume Discount** checkbox selected in the contact record, recalculate the task line item amount using the specified **Discount Percentage** field.
Qualifier	Checks the value in the **Volume Discount** checkbox in the contact record of the vendor.
Sample Values	**Object Definition:** INVC (system object Invoice) **Related Object:** CONT (system object Contact) **Categories of Related Object:** CONT_VNDR (category of Contact)

	Custom Fields of Related Object: volumeDiscount (checkbox)

The following sample is an example of checking values in vendor's contact record.

Java Qualifier: Checking Values in Vendor's Contact Record

```
public class CheckingValueInVendorContactRecord extends CustomCondition<Invoice>
{
    @Override
    public boolean condition(Invoice invoice) {
        // Get invoice service object
        InvoiceService invoiceService = platform.getInvoiceService();

        // Getting old invoice object
        Invoice oldInvoice = invoiceService.readLastSaved(invoice);
        Contact vendor = invoice.getVendor();

        // Return true if the vendor has volume discount set to true and the line
        item count on the invoice has increased
        return vendor.getBooleanFieldValue("CONT_VNDR", "volumeDiscount") &&
        invoiceService.getLineItemCount(invoice) >
        invoiceService.getLineItemCount(oldInvoice);
    }
}
```

Checking Whether Custom Field Has a Value

The following qualifier checks whether a custom field that was previously null has a value, by comparing the value in the old object with the value in the current object.

This qualifier can be used with the action described in Creating Related Task Record.

Business Rule	When the **Trial Date** field in a Matter updates with a value, create a task for the main assignee to complete the executive summary, with the due date 15 days before the trial date.
Qualifier	Checks the value in the **Trial Date** field in a matter record.
Sample Values	**Object Definition:** MATT (custom object Matter) **Categories:** MATT (root category of Matter) **Custom Fields:**

	trialDate (Date)

Java Qualifier: Checking Whether Custom Field Has a Value

```
public class CreatingARelatedTaskRecordQualifier extends
CustomCondition<Project> {
    @Override
    public boolean condition(Project project) {
        // Get old object
        Project oldProject = platform.getProjectService().read(project);

        // Get current trial date
        CalendarDate trialDate = project.getCalendarDateFieldValue("MATT",
        "trialDate");

        // Get old trial date
        CalendarDate oldTrialDate = project.getCalendarDateFieldValue("MATT",
        "trialDate");

        // Return true if the current trial date is not null and the previous
        trial date was null
        return trialDate != null && oldTrialDate == null;
    }
}
```

1.4.3.3 Automated Actions

This section provides samples of automated actions written in Java in the following sections:

- Rule Actions
- Parameterized Actions
- Wizard Page Actions

1.4.3.3.1 Rule Actions

You can use the following samples of automated actions written in Java with rules of type Custom Action:

- Copying Values from Parent to Multiple Child Records
- Creating a History Record
- Updating Added Categories
- Checking and Setting Values in Custom Fields of Type List
- Creating Related Task Record

For parameterized rule action samples, see Parameterized Actions. For wizard page action samples, see Wizard Page Actions.

Copying Values from Parent to Multiple Child Records

The following Java action gets the values of two custom fields of type date in the current object record (custom object Complaint) and copies the values of these fields into date fields in the child records of a particular custom object definition (custom object Served Company).

Business Rule	When the **Service Date** and **Received Date** custom fields in the parent complaint record update, the same values must copy to the corresponding **Service Date** and **Received Date** fields in the child served company records.
Action	Copy the values from the date custom fields in the parent complaint record to the corresponding date custom fields in all of its child served company records.
Sample Values	**Object Definition:** COMP (custom object Complaint) SECO (custom object Served Company, child of Complaint) **Categories:** COMP (root category of Complaint) SECO (root category of Served Company) **Custom Fields:** In Complaint: • serviceDate (Date) • receivedDate (Date) In Served Company: • serviceDate (Date) • receivedDate (Date)

Java Action: Copying Values from Parent to Multiple Child Records

```
public class CopyingValuesFromParentToMultipleChildRecords extends
CustomAction<Project> {
    @Override
    public void action(final Project project) {
        // Run as system user to bypass security
        platform.getUtilityService().runAsSystemUser(new Callable() {
            @Override
            public void call() {
```

```
            // Get service date and received date field values from parent
            CalendarDate serviceDate =
            project.getCalendarDateFieldValue("COMP", "serviceDate");
            CalendarDate receivedDate =
            project.getCalendarDateFieldValue("COMP", "receivedDate");


            // Get a set of child objects for a specific child object
            definition
            List<Project> servedCompanies =
            platform.getProjectService().getChildProjects(project, "SECO");


            for (Project company : servedCompanies) {
                // Set the values from the parent project fields onto each
                child
                company.setCalendarDateFieldValue("SECO", "serviceDate",
                serviceDate);
                company.setCalendarDateFieldValue("SECO", "receivedDate",
                receivedDate);
            }
        }
    });
  }
}
```

Creating a History Record

The following Java action creates a history record and relates it to the current object, which is a contact record. This action would be executed whenever a contact's name is updates.

Business Rule	If the name of a contact modifies, write the following information to the contact History, under the **Data Change** category: • Old name of the contact. • New name of the contact. • Date when the change occurred. • ID of the user who made the change.
Action	Create a new history record with the following information in its fields: **Text**: Contact name changed from <old name> to <new name>. Changed by <user ID>. ***Note:*** *The date of the change was made is automatically recorded in the* ***Created On*** *system field of the history record.* **Default Category**: Data Change

	Parent: Contact record where name change took place. **Entered by**: User who made the change.
Sample Values	**Object Definition:** contact (system object Contact) **Categories:** HIST_DACH (category of History) **Custom Fields:** None

Java Action: Creating a History Record

```
public class CreatingAHistoryRecord extends CustomAction<Contact> {

    @Override
    public void action(final Contact contact) {
        // Run as system user
        platform.getUtilityService().runAsSystemUser(new Callable() {
            @Override
            public void call() {
                // Get old contact
                final Contact oldContact =
                platform.getContactService().readLastSaved(contact);

                // Create a string for history text using old and new values of
                the contact's name
                String historyText = String.format("Contact name changed from %s
                to %s",
                oldContact.getDisplayString(), contact.getDisplayString());

                // Create a history record and related it to the contact
                History history =
                platform.getHistoryService().newHistory(historyText, contact);

                // Set the primary category of the history to Data Change
                history.setPrimaryCategory("HIST_DACH");
            }
        });
    }
}
```

Updating Added Categories

The following Java action removes specific categories from the current object record and adds other categories.

In this example, when the value of the **Pre/Post** custom field in a claim record updates, the corresponding **Pre** or **Post** category is added. Each project can have only one of the two categories at a time. This action automatically sets the categories according to the value in the **Pre/Post** field.

Business Rule	When the **Pre/Post** field value updates, the corresponding **Pre** or **Post** category is added to a Claim project. Each project can have only one of the two categories at a time.
Action	Delete the old category and add the new category.
Sample Values	**Object Definition:** CLAM (custom object Claim) **Categories:** CLAM (root category of Claim) CLAM_POST CLAM_PREE **Custom Fields:** PrePost (List) **Custom Field Value:** PRPO_ROOT_PREE (full tree position of lookup item)

Java Action: Updating Added Categories

```
package com.mitratech.teamconnect.enterprise.api.model.rule;
import com.mitratech.teamconnect.enterprise.api.callable.Callable;
import com.mitratech.teamconnect.enterprise.api.model.Project;
public class UpdatingAddedCategories extends CustomAction<Project> {
    @Override
    public void action(final Project record) {
        // The record must be declared as final if it is going to be referenced
        in a Callable
        // Get an instance of the utility service in order to run code as system
        user
        platform.getUtilityService().runAsSystemUser(new Callable() {
            public void call() {
                // Getting the value of the Pre/Post custom field
                String newPrePostKey = record.getProjectFieldValue("CLAM",
                "PrePost").getUniqueKey();
                if (newPrePostKey.equals("PRPO_ROOT_PREE")) {
                    record.removeCategory("CLAM_POST");
```

```
                record.addCategory("CLAM_PREE");
            } else {
                record.removeCategory("CLAM_PREE");
                record.addCategory("CLAM_POST");
            }
        }
    });
  }
}
```

Checking and Setting Values in Custom Fields of Type List

The following Java action automatically sets the value of one custom field according to the value selected in another custom field.

Business Rule	If the value in the **Ice Cream** custom field is Chocolate Cake, Chocolate Chip Cookie Dough, Moose Tracks, or Chocolate Peanut Butter, select **Good** in the **Tastiness** custom field. Otherwise, select **Okay**.
Action	Set the **Tastiness** custom field value to **Good** or **Okay**.
Sample Values	**Object Definition:** project (custom object) **Custom Fields:** icecream (List) Tastiness (String) **Custom Field List Values:** In custom lookup table icecream: • ICRM_ROOT_CHOC_CAKE • ICRM_ROOT_CCHP_CDGH • ICRM_ROOT_MOOS_TRKS • ICRM_ROOT_CHOC_PBTR In custom lookup table Tastiness: • Good • Okay

Java Action: Checking and Setting Values in Custom Fields of Type List

```
public class IceCreamCondition extends CustomCondition<Project> {
```

```
    @Override
    public boolean condition(Project project) {
        // Getting the value of the ice cream custom lookup field
        String flavor = project.getLookupFieldValue("icecream").getKey();
        /* Check whether the value of the field is one of the four ice cream
        flavors that contains chocolate */
        if (flavor.equals("CHOC_CAKE") || flavor.equals("CCHP_CDGH") ||
        flavor.equals("MOOS_TRKS") || flavor.equals("CHOC_PBTR")) {
            // If value is one of the four values, set "Tastiness" custom field to
            Good.
            project.setTextFieldValue("Tastiness", "Good");
        } else {
            // If value is NOT one of the four values, set "Tastiness" custom
            field to Okay.
            project.setTextFieldValue("Tastiness", "Okay");
        }

        /* This code can also be done without using keys, such as getting the
        value of a text field which is not going to have a key associated with
        it. */

        // Getting the value of the ice cream custom lookup field
        String flavorText = project.getTextFieldValue("icecream");
        /* Check whether the value of the field is one of the four ice cream
        flavors that contains chocolate */
        return (flavorText.equals("CHOCOLATE CAKE") ||
        flavorText.equals("CHOCOLATE CHIP COOKIE DOUGH") ||
        flavorText.equals("MOOSE TRACKS") || flavorText.equals("CHOCOLATE PEANUT
        BUTTER"));
    }
}
```

Creating Related Task Record

The following Java action creates a task record, relates it to the current object, and sets the default category. The due date of the task is set to 15 days before the date in a custom field in the current object.

Business Rule	When the **Trial Date** field in a matter record updates with a value, create a task for the main assignee to complete the executive summary, with the due date 15 days before the trial date.
Action	Create a task with the following information in its fields: **Parent Project:**

	The Matter project **Assignee:** Main project assignee **Due On:** 15 days before the trial date **Description:** Complete executive summary **Default Category:** Trial Preparation
Sample Values	**Object Definition:** MATT (custom object Matter) **Related Object Created:** Task (system object) **Categories:** MATT (root category of Matter) TASK_TRPR (Task category) **Custom Fields:** trialDate (Date field in Matter, created under the root category)

Java Action: Creating Related Task Record

```
public class CreatingRelatedTaskRecord extends CustomAction<Project> {
    @Override
    public void action(final Project project) {
        // Run as system user to bypass security
        platform.getUtilityService().runAsSystemUser(new Callable() {
            @Override
            public void call() {
                // Creating task and assigning it to main assignee of current
                object
                Task task = platform.getTaskService().newTask("Complete executive
                summary", project.getMainAssignee().getUser());
                CalendarDate trialDate = project.getCalendarDateFieldValue("MATT",
                "trialDate");

                // Set the due on date to fifteen days before the trial date
                Calendar dueOn = Calendar.getInstance();
                dueOn.setTime(trialDate);
```

```
                dueOn.add(Calendar.DAY_OF_YEAR, -15);
                task.setDueDate(new CalendarDate(dueOn));

                // Set primary category to trial preparation
                task.setPrimaryCategory("TASK_TRPR");
            }
        });
    }
}
```

1.4.3.3.2 Parameterized Actions

Rule parameters appear in the user interface and allow the administrator to enter values for the rule. Having parameters in rules is useful when certain values in a qualifier or an action may change and you prefer to have the administrator change the value through the user interface rather than modify the file. Parameterized rules are also useful because you can use the same file for multiple rules that each require different values.

Parameters for a wizard page action work in the same way as a rule, expect they appear in the wizard.

This section includes the following parameterized action samples:

- Specifying Due Date for Task Record Using Parameter
- Creating Project with Related History Using Parameters

Specifying Due Date for Task Record Using Parameter

The following sample automatically creates a task record when the user is saving a claim record for the first time. The due date of the task is set automatically according to the number of days specified by the administrator in the parameter, which appears as a field on the Rule screen in the user interface.

Business Rule	When the **Police Report Issued** checkbox is selected in a Claim, the system creates a task.
Action	Creates a task to obtain a copy of the report with the following information in its fields: **Parent Project**: the Claim project. **Assignee**: Main project assignee. **Description**: Obtain a copy of the police report. **Default Category**: Follow-up:FNOL. **Parameter**: A Number field with the name "numberOfDaysAfter" and the label "The number of days between the date the claim was created and the due date"

	Due On: Parameter numberOfDaysAfter.
Sample Values	**Object Definition:** Claim or any custom object definition **Rule Parameter Created:** numberOfDaysAfter (Integer) **Related Object Created:** TASK (system object Task) **Categories in Related Object:** TASK_FLUP_FNOL **Custom Fields:** None

Java Parameterized Action: Specifying Due Date for Task Record Using Parameter

```
public class SpecifyingDueDateForTaskRecord extends CustomAction<Project> {

    @Override
    public void declareParameters() {
        parameters.addNumberParameter("numberOfDaysAfter", "The number of days
        between the date the claim was created and the due date",
        Long.valueOf(1));
    }

    @Override
    public void action(final Project project) {
        // Run as system user to bypass security
        platform.getUtilityService().runAsSystemUser(new Callable() {
            @Override
            public void call() {
                // Get the value of the rule parameter
                int numberOfDaysAfter =
                parameters.getNumberParameterValue("numberOfDaysAfter").intValue()
                ;


                // Create a task and assigning it to the main assignee of the
                project
                Task task = platform.getTaskService().newTask("Obtain a copy of
                the police report", project.getMainAssignee().getUser());


                // Relate the task to the project
                task.setProject(project);
```

```
            // Set the due date based on the number of days specified in the
            rules parameter
            Calendar dueOn = Calendar.getInstance();
            dueOn.setTime(project.getCreatedOn());
            dueOn.add(Calendar.DAY_OF_YEAR, numberOfDaysAfter);

            // Set the primary category of the task
            task.setPrimaryCategory("TASK_FLUP_FNOL");
        }
    });
  }
}
```

Creating Project with Related History Using Parameters

The following Java action uses multiple parameters. Based on these parameter values, the action creates a Package Request custom object record. It then determines whether to create a related history record based on a boolean parameter, and if so, creates the history record with the correct values.

Business Rule	When the value in the **Accident State** custom field in a new Claim is New York, a letter Package Request project must be created to generate the appropriate notices for the insured and claimant.
Action	Create a Package Request with the following information: **Requested for**: Claim record. Parameter 1: Text field named "categories" and labeled "Type the full tree position of the Letter Package Request categories, separated by commas:" Parameter 2: Number field with the name "numberOfDaysAfter" and the label "Type the number of days from the moment the rule is triggered to specify the date when the letters must be sent out:" Parameter 3: Check Box field with the name "isAuditCreated" and the label "Specify whether an audit file should be created for each request:" Using Parameter 1, add the specified categories to the package request record. **Send Date**: Parameter 2. **Create Audit File**: Parameter 3.

	• If yes—create a history record with following data in its fields: **Text**: NY Package request generated. **Default Category**: Audit **Parent**: the Claim project. **Entered by**: TeamConnectAdmin. • If no—no other records are created.
Sample Values	**Object Definition:** Claim or any custom object definition **Related Objects Created:** PKRE (custom object Package Request) TNHistory (the audit file) **Wizard Parameters:** categories (Text) numberOfDaysAfter (Number) isAuditCreated (Check Box) **Categories:** PKRE (root category of Package Request) HIST_AUDI (History category) **Custom Fields:** In PKRE: requestedFor (custom field of type Custom Object, pointing to the same custom object definition as the one for which the rule is written) sendDate (Date)

Java Parameterized Action: Creating Project with Related History Using Parameters

```
public class CreatingProjectWithRelatedHistory extends CustomAction<Project> {


    @Override
    public void declareParameters() {
        parameters.addTextParameter("categories", "Full tree position of the
        Letter Package Request categories, separated by commas", null);
        parameters.addNumberParameter("numberOfDaysAfter", "Number of days
        between the rule triggering and the date the letters must be sent out",
        Long.valueOf(1));
```

```
        parameters.addBooleanParameter("isAuditCreated", "Whether an audit file
        should be created for each package request", null);
    }

    @Override
    public void action(final Project project) {
        // Get the parameter values
        final int numberOfDaysAfter =
        parameters.getNumberParameterValue("numberOfDaysAfter").intValue();
        final boolean isAuditCreated =
        parameters.getBooleanParameterValue("isAuditCreated");
        final String[] categories =
        parameters.getTextParameterValue("categories").split(Pattern.quote(","));

        // Run as system user to bypass security
        platform.getUtilityService().runAsSystemUser(new Callable() {
            @Override
            public void call() {
                // Creating new package request
                Project packageRequest =
                platform.getProjectService().newProject("PKRE");

                // Setting value of Requested For custom field to the current
                project object
                packageRequest.setProjectFieldValue("PKRE", "requestedFor",
                project);

                for (String category : categories) {
                    packageRequest.addCategory(category);
                }

                Calendar sendDate = Calendar.getInstance();
                sendDate.setTime(new Date());
                sendDate.add(Calendar.DAY_OF_YEAR, numberOfDaysAfter);
                packageRequest.setDateFieldValue("PKRE", "sendDate",
                sendDate.getTime());

                if (isAuditCreated) {
                    // Create history with primary category set to audit
                    History history =
                    platform.getHistoryService().newHistory("Package request
                    generated", project);
                    history.setPrimaryCategory("HIST_AUDI");
                }
            }
        });
```

```
    }
}
```

1.4.3.3.3 Wizard Page Actions

The construction of a wizard action is almost the same as the construction of a rule action, except the action identifies and uses parameters in different ways. For details about wizard page actions, see Wizard Automated Actions.

You can use the following examples of wizard page actions in Java. These actions execute when the wizard user clicks next to go to the next wizard page.

- Creating a Child Project
- Setting User with Specific Role in Project as Task Assignee
- Filtering Users for Manual Selection
- Automatically Selecting Project Assignees

Creating a Child Project

Automated actions in wizards can create related records for a main record. While you can also create related records through a template, by creating related records through a page action, you can specify that the related record only create when certain conditions exist.

In this example, when the **Any Injuries?** checkbox is selected in a wizard, the system creates a child BI Claim. The following wizard action automatically creates a child BI Claim project.

What User Does On Wizard Page	User selects the **Any Injuries?** checkbox (a wizard parameter of type Boolean) in the First Notice of Loss wizard
Wizard Page Action	Create a child Claim with the default category Bodily Injury
Sample Values	**Object Definition:** CLAM (custom object Claim) **Categories:** CLAM_BOIN **Custom Fields:** None **Wizard Parameters:** anyInjury (boolean)

Java Wizard Action: Creating a Child Project

```
public class CreatingAChildProject extends CustomAction<Project> {
```

```
    @Override
    public void action(final Project project) {
        // Get wizard parameter
        final boolean isAnyInjury =
        parameters.getBooleanParameterValue("anyInjury");

        // Run as system user to bypass security
        platform.getUtilityService().runAsSystemUser(new Callable() {

            @Override
            public void call() {
                // Create new claim as a child of the project
                Project claim = platform.getProjectService().newProject("CLAM");
                claim.setParentProject(project);

                // Add BI category as primary
                claim.setPrimaryCategory("CLAM_BOIN");
            }
        });
    }
}
```

Setting User with Specific Role in Project as Task Assignee

The wizard action automatically assigns the task to the first user who has the **Proposed Attorney** assignee role in the matter.

What User Does On Wizard Page	User adds an assignee to the matter wizard.
Wizard Page Action	Checks the role of the assignee, and if a user has the **Proposed Attorney** role, add this user as the assignee to the template-created task record associated with the wizard.
Sample Values	**Object Definition:** MATT (custom object Matter) **Assignee Roles:** MATT_PRAT (Proposed Attorney) **Categories:** None

	Custom Fields: None **Wizard Parameters:** project (project) **Related Object:** task (a Task)

Java Wizard Action: Setting User with Specific Role as Assignee in Task

```
public class SettingUserWithRoleAsAssigneeInRelatedTask extends
CustomAction<Task> {

    @Override
    public void action(final Task task) {
        // Retrieve project wizard parameter
        final Project project = parameters.getProjectParameterValue("project");

        // Run as system user to bypass security
        platform.getUtilityService().runAsSystemUser(new Callable() {

            @Override
            public void call() {
                // Get list of project assignees
                List<ProjectAssignee> assignees = project.getAssignees();

                for (ProjectAssignee assignee : assignees) {
                    // Assign the project assignee with the 'Proposed Attorney'
                    role
                    if (assignee.getAssigneeRole().equals("MATT_PRAT")) {
                        platform.getTaskService().reassign(task,
                        assignee.getUser());
                        break;
                    }
                }
            }
        });
    }
}
```

Filtering Users for Manual Selection

The following wizard page action assigns a list of users to a project when they have a certain primary category in their contact records.

What User Does On Wizard Page	User selects **Bodily Injury** as the default category for a claim record.
Wizard Page Action	Selects all BI adjusters and lists them as assignees on the project.
Sample Values	**Object Definition:** CLAM (custom object Claim) **Assignee Roles:** ADJU (BI Adjuster) **Categories:** CONT_ADJU_BIAD (category of Contact) **Custom Fields:** None **Wizard Parameters:** None

Java Wizard Action: Filtering Users for Manual Selection

```
public class FilteringUsersForManualSelection extends CustomAction<Project> {

    @Override
    public void action(final Project project) {

        // Run as system user to bypass security
        platform.getUtilityService().runAsSystemUser(new Callable() {
            @Override
            public void call() {
                List<User> users = platform.getUserService().getAllUsers();

                for (User user : users) {
                    // If the user's contact's primary category is BI Adjuster, add
                    to adjusters
                    if
                    (user.getContact().getPrimaryCategory().getTreePosition().equal
                    s("CONT_ADJU_BIAD")) {
                        project.addAssignee("ADJU", user);
                    }
                }
            }
        });
```

```
    }
}
```

Automatically Selecting Project Assignees

The following wizard page action first checks whether the claim record created by the wizard has the category BI. It then gets a list of users by checking the default category (Bodily Injury) and the value in the **Is available** checkbox in the user's contact record. The action also assigns the first available BI Adjuster to the claim and then clears the **Is available** checkbox.

What User Does On Wizard Page	User adds the **Bodily Injury** category to a claim record created through a wizard.
Wizard Page Action	Assigns the first available BI Adjuster to the claim by checking the default category (Bodily Injury) and the value in the **Is available** checkbox in the user's contact record. Then, clears the **Is available** checkbox.
Sample Values	**Object Definition:** CLAM (custom object Claim) **Assignee Roles:** CLAM_ADJU (Claim Adjuster) **Categories:** CLAM_BDIN (category in Claim) CONT_ADJU_BIAD (category in Contact) CONT_ADJU (category in Contact) **Custom Fields:** isAvailable (checkbox in Contact) **Wizard Parameters:** None

Java Wizard Action: Automatically Selecting Project Assignees

```
public class AutomaticallySelectingProjectAssignee extends CustomAction<Project>
{


   @Override
   public void action(final Project project) {
       try {
          platform.getUtilityService().runAsSystemUser(new
          CallableWithException<Exception>() {
```

```
            @Override
            public void call() throws Exception {
                // Check whether BI Category has been added to the claim record
                boolean isBI = project.hasCategory("CLAM_BDIN");

                // Get all users
                List<User> users = platform.getUserService().getAllUsers();

                for (User user : users) {
                    // Check primary category and value of Is Available field on
                    each user's contact
                    Contact contact = user.getContact();
                    if (contact.getPrimaryCategory().equals("CONT_ADJU_BIAD") &&
                    contact.getBooleanFieldValue("CONT_ADJU", "isAvailable")) {
                        // Add the user as a project assignee with the role BI
                        Adjuster
                        project.addAssignee("CLAM_ADJU", user);

                        // Make the contact no longer available
                        contact.setBooleanFieldValue("CONT_ADJU", "isAvailable",
                        false);
                    }
                }
            }
        });
    } catch (Exception e) {
        // Log a message with the exception attached
        logWarn("The category CLAM_BDIN was not added to Claim", e);
    }
  }
}
```

1.4.3.4 Searching Samples

You can use the following searching examples written in Java in your custom code. In these examples, the values in the current object are used unless otherwise stated.

You can find the following search samples in this section:

- Searching for Child Accounts with Certain Posting Criteria
- Getting a List of Tasks Completed within a Certain Time Period

Searching for Child Accounts with Certain Posting Criteria

The following code searches for all accounts with the same parent account and a specific project set as their posting criteria.

Search Requirements	Find all child accounts of the Matter XYZ Budget account that have a specific litigation record with the unique ID 7/15/2005-6548 set as their posting criteria.
Search Criteria	AND statement with the following qualifiers: • Parent account Matter XYZ Budget. • Litigation project with the unique ID 7/15/2005-6548 set as posting criterion.
Sample Values	**System Field Attributes:** NAME (of the parent account) NUMBER_STRING (ID of the project) **Categories:** None **Custom Field Attributes:** None

Searching for Child Accounts with Certain Posting Criteria

```
// An example of building more complicated field paths
AccountService accountService = platform.getAccountService();


// Search by the name of the parent account
FieldPath parentAccountName = new
FieldPath(Account.PARENT_ACCOUNT).add(Account.NAME);
StringCriterion parentAccountNameCriterion = new
StringCriterion(parentAccountName).equalTo("Matter XYZ Budget");


// Search by the number string of the project
FieldPath projectNumberString = new
FieldPath(AccountPostingCriteria.PROJECT).add(Project.NUMBER_STRING);
StringCriterion projectNumberStringCriterion = new
StringCriterion(projectNumberString).equalTo("7/15/2005-6548");


// Join the criteria together
SearchCriteria criteria = new SearchCriteria(parentAccountNameCriterion,
projectNumberStringCriterion);


List<account> accounts = accountService.search(criteria);
```

Getting a List of Tasks Completed within a Certain Time Period

The following code gets a list of all tasks completed within the specified time period.

Search Requirements	Find all tasks John Smith completed within the month of June 2005.
Search Criteria	AND statement with the following qualifiers: • User object of the task assignee. • Tasks that are marked as completed. • The start date of the specified period, 06/01/2005. • The end date of the specified period, 06/30/2005.
Sample Values	**System Field Attributes:** CURRENT_ASSIGNEE COMPLETED_DATE **Categories:** None **Custom Field Attributes:** None

Searching for All Tasks Completed within a Certain Period

```
TaskService taskService = platform.getTaskService();

// Find tasks only for the given user
UserCriterion userCriterion = new
UserCriterion(Task.CURRENT_ASSIGNEE).equalTo(assignee);

// Find tasks completed after the given start date and before the given end date
DateCriterion dateCriterion = new
DateCriterion(Task.COMPLETED_DATE).between(6/01/2005, 6/30/2005);

// Join together the two criteria, using 'and' logic by default
SearchCriteria searchCriteria = new SearchCriteria(userCriterion,
dateCriterion);

// Execute search and return results
List<Task> tasks = taskService.search(searchCriteria);
```

1.5 Web Services Help

Welcome to the *TeamConnect® Enterprise Web Services Help*.

Common Web Service Functions	Repository Methods
Abstract Classes	Code Samples

1.5.1 Getting Started

Welcome to TeamConnect Web Services. To get started building application interfaces with the TeamConnect system, follow these steps:

1. Get comfortable with high-level concepts in this chapter.
2. Decide which features of TeamConnect® Enterprise you'd like to use. Take a look through the section Supported Operations.
3. Use the Web Service API Reference chapter to get a further idea of the types of requests you can make per repository or TeamConnect Web Service.
4. Use your 3rd party SOAP toolkit to generate client-side source files and API for the TeamConnect Web Service .WSDL and schema (.XSD) files provided.

   The resulting client-side source files (in the programming language you will use for your client application) will provide more specific syntax for the repository, method, parameter, return names/values to use in your client application.
5. Create your client application.

1.5.1.1 Overview

TeamConnect Web Services provide repositories (for ex. ContactRepository) that are interfaces to make Web Service requests. The result of a TeamConnect Web Service request will generally be data insertion to, update to, retrieval from, or deletion from a TeamConnect database.

To work with TeamConnect records supported by Web Services, you build TeamConnect Web Service client programs that use the TeamConnect Web Service API. You'll need to use a SOAP toolkit (such Apache CXF) that connects to and interacts with TeamConnect® Enterprise.

You can write a client program in the programming language of your choice. Write a client program to send a request to one of the TeamConnect Web Services such as the DocumentRepository or ContactRepository.

You don't need to install the TeamConnect Web Service API. You only need to install software for your programming language and the SOAP toolkit you will use to write client programs. For example, if you will write client programs in Java, you need to install the Java development kit and a SOAP toolkit such as Apache CXF.

TeamConnect Web Service operations are defined in WSDL (Web Services Definition Language) files. The TeamConnect Web Service source files (.WSDL and .XSD) are provided in the

TeamConnect installation package. You need to host the .WSDL and .XSD files on a web server or application server. To connect to a TeamConnect Web Service, you need to declare the URL to the .WSDL files in your client programs. This guide provides a format standard for the .WSDL URLs. Refer to your SOAP toolkit documentation for pre-defined functions provided to connect to a TeamConnect Web Service.

After establishing a connection to the WSDL, the client program can send requests to the TeamConnect Web Service to perform any operations supported by the TeamConnect Web Service. For example, after connecting to the ContactRepository, a client program could send an insertContact request to add a contact record.

A client program sends a SOAP request which the TeamConnect Web Service processes and sends a response. The request and response messages are XML files with a header and body.

For TeamConnect Web Service requests, the header must include a valid TeamConnect user name and password (also note that the user should belong to a group with sufficient rights to perform the requested operations). The header also lists classes that are referred to in the message. The message body specifies the requested operation with related parameters.

The WSDL source files provided with TeamConnect define the requests that a Web Service can process. The WSDL file includes the operations that a TeamConnect Web Service can perform, required parameters per operation, and the response per operation. SOAP header elements and errors thrown are also described in the WSDL file.

Typically before you begin to use TeamConnect Web Services, you need to download a SOAP toolkit or framework (such as Apache CXF) that interprets WSDL files, and also encodes and decodes XML requests and responses. When a TeamConnect Web Service receives a request, it sends an XML response. The SOAP toolkit will parse the response in a format appropriate to the programming language of the client program.

In addition, the SOAP toolkit generates stubs or client proxies that know how to locate TeamConnect® Enterprise. Usually one stub or client proxy is generated for each Web Service and your client application is written against the stubs.

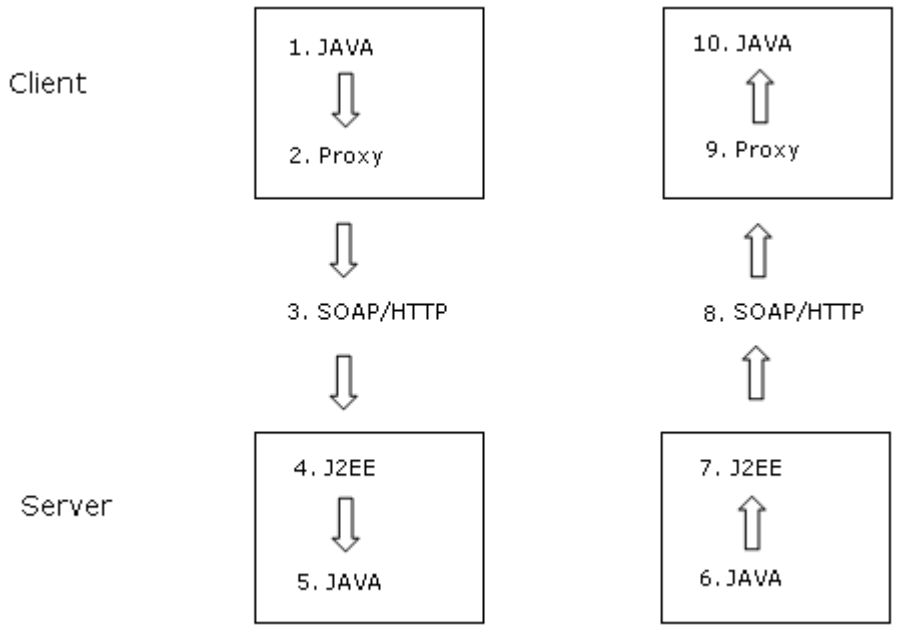
The toolkit you would use depends on the programming language you are using. Depending on the toolkit you use, you may not need to write any XML or you may write some of the XML messages. Note that the TeamConnect Web Service API uses document/literal style SOAP 1.1 (not rpc/encoded style) and WSDL 1.1.

Note: *When using TeamConnect Web Service functions, note that TeamConnect Rules associated with the operation you are performing such as update or insert will apply (for example, field validations).*

Important: *Samples in this guide use JAVA syntax (specific to the Apache CXF wsdl2java-generated source). If you are using a different programming language, you must get a corresponding 3rd party SOAP toolkit to convert the provided .WSDL and .XSD files to the appropriate source code. If you are using a programming language other than JAVA and SOAP toolkit other than Apache CXF, you should feel comfortable with the SOAP toolkit vendor's support.*

Note: *The code samples provided in this guide are not complete applications. You may need to refer to 3rd party software documentation for the tool you use to generate client-side code for how*

to perform authentication and how to create client proxies for repositories. For example, see Putting Components Together in a Client Application for more information.



Data Flow Diagram for JAVA Application Using Web Service API

1.5.1.1.1 Supported Operations

The following table summarizes supported Web Service operations by TeamConnect object type.

Supported Web Service Operations

TeamConnect Object Type	Supported Operations
Accounts	• insert • read • readChildAccounts • readRecentlyViewed • update • readByCriteria • activateAccount • deactivateAccount • allocateMoney • transferMoney

	• withdrawMoney • delete
Appointments	• insert • read • update • readByCriteria • readRecentlyViewed • delete
Contacts	• insert • read • update • readByCriteria • delete • readRecentlyViewed
Documents	• insert • update • read • readByPath • readChildDocuments • readChildDocumentForName • readDocumentFolderForDocumentOwner • readByCriteria • checkIn • checkOut • undoCheckout • copy • createShortcut • createSubFolder • createHyperlinkDefaultCategory • createHyperlink

	• move • revert • delete • readRecentlyViewed • setAsRecentlyViewed
Expenses	• insert • read • update • readByCriteria • post • void • delete • readRecentlyViewed
Group Accounts	• insert • read • update • readByCriteria • delete • readRecentlyViewed
History records	• insert • read • update • readByCriteria • delete • readRecentlyViewed
Invoice	• insert • read • update • readByCriteria

	• post • void • delete • readRecentlyViewed • adjustInvoiceHeaders • readActiveApprovals • readCompletedApprovals • readInvoiceApprovalsPendingOnPost
Involved	• insert • read • readInvolvedsForProject • readInvolvedsByCriteria • update • delete
Projects	• insert • read • update • readChildProjectsForEntityType • readByCriteria • changePhase • delete • readRecentlyViewed • readProjectEntityTypes • readProjectIntegrationSearches • readProjectsUsingSearch
Tasks	• insert • read • update • readByCriteria • post

	• void • reassign • delete • readRecentlyViewed
User Accounts	• insert • read • update • readByCriteria • delete • readRecentlyViewed

What is Not Supported

See Operations Not Supported In Web Services for a list of functions that need to be performed by using the TeamConnect application user interface (UI).

1.5.1.2 Key Concepts

This section provides general concepts or conventions to understand before you start to use TeamConnect Web Services.

1.5.1.2.1 Repository Classes

Before you begin, you should understand that each TeamConnect object type is associated with a repository class (for example, ContactRepository). Methods are provided per repository for common TeamConnect functions. For example:

- insertContact (ContactCreate contact)
- updateContact (ContactUpdate contact)
- readContact (String uniqueKey, List properties)
- readContactsByCriteria (SearchCriteria criteria, Int limit, List properties)
- deleteContact (String uniqueKey)

Note: *For a list of available properties that can be retrieved per record, see the elements listed in the original types.xsd file under the webservices directory. For example, the complexTypes, person, has a list of elements whose names are available properties to read or search from TeamConnect .*

One HTTP request results from each repository class call. However, there are multiple subclasses associated with each TeamConnect object type (or repository class) that you use to store data (such as field values) when inserting (creating), updating, reading, or deleting TeamConnect records.

In a client program you instantiate a repository class once but you can use that repository instance multiple times to work with multiple records. You use the type classes or record unique keys as parameters to identify which record you are working with per request.

1.5.1.2.2 Selective Record Updating and Reading

To maximize performance, methods to update type classes (TeamConnect records) and read type classes (TeamConnect records) allow you to specify the properties you want to update or read.

When you update a type class (TeamConnect record) using a Web Service, you identify the record to update by passing its unique key. The update request only updates your specified properties. Unlike a record update made by a TeamConnect rule, the record does not need to be retrieved first in order to change a property. This feature offers better performance through fewer HTTP requests and smaller data packets.

When you read a type class (TeamConnect record) using a Web Service, you must specify which properties to return. Only the specified property values are returned with a corresponding object. This feature improves performance through smaller data packets.

Note: When a record is read by a TeamConnect rule, the returned object includes all of its property values.

Important: When reading records, any properties you do not include in the List<String> properties parameter may be returned with a null or 0 (zero) value. For example, if you are reading a contact record and specify three property values to return out of 24 properties available, then the remaining 21 properties will be returned with either null or 0 values. Note that you should ignore the null and 0 values for properties you did not explicitly retrieve.

1.5.1.2.3 Data Types

After you use a 3rd party tool (for example, Apache CXF wsdl2java) to generate client-side source code, you can view the resulting classes that provide information about data types in the corresponding directory: ...\com\mitratech\teamconnect\webservice\type

Dates and Time Zone Independence

When working with date fields (properties of type Date), dates can be either time-zone independent or time-zone dependent. You need to refer to the types.xsd and find the property's corresponding element type. If the element type is "dateTime" then the date value is time-zone dependent. If the type is "date" then the date value is time-zone independent.

1.5.1.2.4 Request and Response Formats

Each repository method (request) call in your client application will be translated by the client proxy (provided by the SOAP toolkit) into an XML message with a SOAP wrapper. Each repository method results in one SOAP request over HTTP to the Web Service (TeamConnect) server.

Note: *Each SOAP request requires authentication to the TeamConnect system. See SOAP Message Security Header for more information.*

1.5.1.2.5 Error Handling

There are a few types of errors you may receive.

- Permission Denied errors will result if the user ID provided during authentication doesn't have sufficient access rights to perform the requested functions.
- Client-side errors. For example, an error would result if you tried to create a record but didn't populate one of its required fields.
- Server-side errors. After you generate client-side source code from the WSDL files, the WebServiceFaultDetail class provides a property, detailedErrorMessage, that is a wrapper for server-side error messages.

  For server-side errors, the server throws the error, which is wrapped in an .XML message. This message is returned to the client and translated to a .JAVA object (or the appropriate programming language object equivalent) with the exception.

1.5.1.2.6 Record Unique Keys

Note: *This section describes the current convention of a record's unique key. This definition may change in the future.*

When updating, reading, or deleting records, you need to pass the desired record's unique key. The unique key format is like: TC_Object_Definition_Unique_Code + "_" + TC_record_primary_key (for example, an invoice's unique key might look like **INVC_3055**).

Note: *The unique key value is not stored directly in the TeamConnect database. It is constructed by two values stored separately in the database.*

If you don't know a record's unique key and want to read or retrieve its properties, you can alternatively use the read*X*ByCriteria request. For example, to read a contact you can use the contactRepository readContactsByCriteria request. You can either specify the properties to read in the target record directly as a readContactsByCriteria parameter or you can just retrieve the uniqueKey property in the search results and pass this value in the readContact request. Typically you will search or use the read*X*ByCriteria request to get a record's unique key. Then you can use the unique key for update, read, or delete operations.

1.5.1.3 Requirements

- A TeamConnect® Enterprise instance
- A valid TeamConnect® Enterprise user ID and password. This user must have sufficient rights to perform operations you plan to use through membership in a group with corresponding rights. For more information, see Account Administration for group record rights (global user rights, global admin rights, global setup rights) and group category and custom field rights.

For projects/matters (custom objects), you will need to manually grant access rights to your group per custom object under the Group **Record Rights** and **Category Rights** areas.

- TeamConnect Web Service source files (.WSDL and .XSD files), located at the following installation path (after running the TeamConnect Installer):

  C:\Program Files\Mitratech\TeamConnect *\utilities\webservices\

  (where * would be replaced by the current release version, for example,

  C:\Program Files\Mitratech\TeamConnect 3.4 SP1\utilities\webservices\ )

- An SDK for the programming language you want to use.

  If you are creating a Java client application, you must use a JDK version that is supported by TeamConnect. Refer to the TeamConnect Release Notes for supported versions.

- A third-party SOAP toolkit to generate the Web Service client API and code from the Web Service WSDL files. For example, Apache CXF. The SOAP toolkit should also provide the framework for creating a client proxy.

  Note: *Samples provided in this guide are based on client-side JAVA code generated using Apache CXF 2.1. If you are using a different tool to generate client-side code, you need to reference the resulting source files.*

1.5.1.3.1 Generating Client Source-code and API

This section describes the generation of the Web Service Client API and source-code using your SOAP toolkit.

Note: *If you will write a JAVA client program and use the Apache CXF toolkit/framework, then skip to the bottom of this section for information about locating the client API.*

Depending on which programming language and SOAP toolkit you plan to use, you will need to use the toolkit to convert the TeamConnect Web Service .WSDL and .XSD files to client-side code (for example .JAVA source). This code serves as the Client API for the TeamConnect Web Service classes, methods, properties, and return values you would write to in your client program.

Copy the TeamConnect Web Service files (.WSDL and schema or .XSD) from the TeamConnect installation path at C:\Program Files\Mitratech\TeamConnect *\utilities\webservices\ (where * is the version number) to a different working directory (where the path does not include spaces or special characters). The following is a list of TeamConnect Web Service files:

- account-repository.wsdl
- account-repository-schema.xsd
- appointment-repository.wsdl
- appointment-repository-schema.xsd
- contact-repository.wsdl
- contact-repository-schema.xsd
- document-repository.wsdl

- document-repository-schema.xsd
- expense-repository.wsdl
- expense-repository-schema.xsd
- group-account-repository.wsdl
- group-account-repository-schema.xsd
- history-repository.wsdl
- history-repository-schema.xsd
- invoice-repository.wsdl
- invoice-repository-schema.xsd
- involved-repository.wsdl
- involved-repository-schema.xsd
- project-repository.wsdl
- project-repository-schema.xsd
- repository-fault-schema.xsd
- task-repository.wsdl
- task-repository-schema.xsd
- types.xsd
- user-account-repository.wsdl
- user-account-repository-schema.xsd

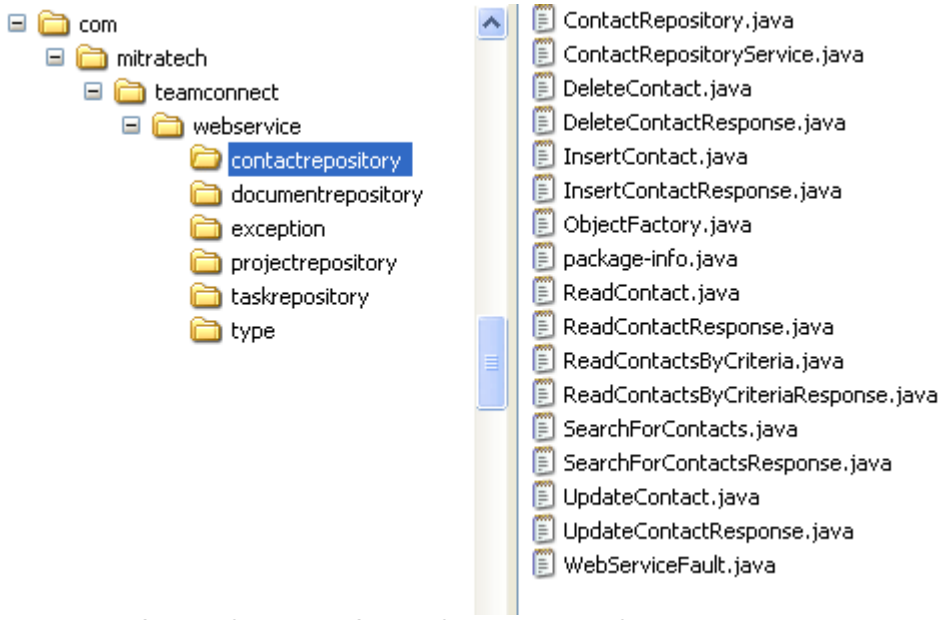
Use the tool provided in the 3rd party SOAP toolkit to convert the .WSDL files to the source code corresponding to the programming language you plan to code.

After generating the client-side source code, you must compile the code. Then package the binaries into a .JAR file (or appropriate package for your programming language, where the root directory should be \com).

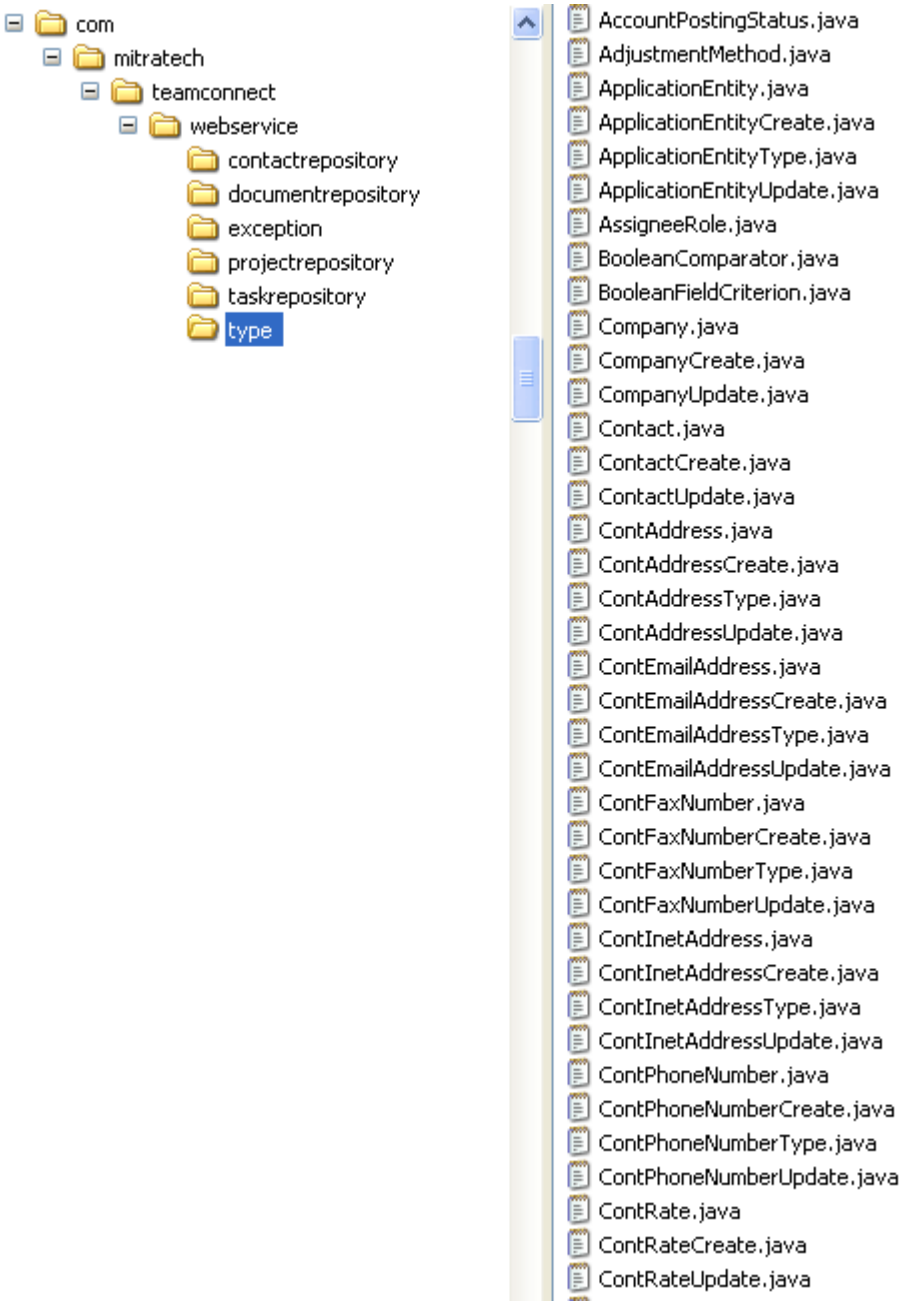
You need to add the resulting .JAR file (or appropriate package for your programming language) to your CLASSPATH environment variable (or appropriate environment variable for your programming language) before you can compile a custom application.

Client Source Code for JAVA and Apache CXF

For example, after generating client-side source code from the contact-repository.wsdl file, a directory structure like the following results:



Sample Client-side Source Code Directory (Contact Repository)



Sample Client-side Source Code Directory (Types)

Continuing with the example of contacts, find the objects that you will typically use in the following generated client-side source code directories:

\com\mitratech\teamconnect\webservice\contactrepository

\com\mitratech\teamconnect\webservice\type

Client API for JAVA and Apache CXF

If you plan to write a JAVA client program and use the Apache CXF toolkit/framework, then you only need to generate the client-side code. You do not need to compile the code because the TeamConnect Installer provides the client API in the form of a .JAR file located in the installation path:

C:\Program Files\Mitratech\TeamConnect *\utilities\webservices\teamconnect-webservice-client-**.jar

(where * and ** will be the current version of TeamConnect)

1.5.1.4 Creating an Application

The following section describes components required for a client application.

- Client Proxy
  - Reference to Client API path
  - Web Service URL
- Security Header (Interceptor)
- Client Application

1.5.1.4.1 Client Proxy

The 3rd party SOAP toolkit should provide a class that creates a client proxy to translate the programming language used in your client application to the XML message format defined in the WSDL files. In your client application, you need to instantiate the given client proxy.

Client API Path

For each client proxy you also need to reference the class name of the client API that will be used. You can do this by passing the target entity name to populate the resulting client API qualified class name. For example, you'd construct the string for the class path to a client interface like :

"com.mitratech.teamconnect.webservice." + entityName.toLowerCase() + "repository." + entityName + "Repository"

(where depending on the repository type you are working with, you would pass in the corresponding entityName, such as Contact)

For more a more detailed example, see the Client Proxy Sample Code section of the Client Application Components (Java/Apache CXF) appendix.

Web Service URL

For each client proxy you also need to reference the URL of the TeamConnect Web Service that will be used. You can do this by passing the target entity name to populate the resulting TeamConnect Web Service URL. For example, you'd construct the string for a TeamConnect Web Service URL like :

"http://" + SERVER + ":" + PORT + "/" + APP NAME + "/webservice/" + StringUtils.uncapitalize(entityName) + "Repository"

(where depending on the repository type you are working with, you would pass in the corresponding entityName, such as Contact)

The format of the resulting Web Service URL would be like:

http://localhost:8080/teamconnect/webservice/contactRepository

For more a more detailed example, see the Client Proxy Sample Code section of the Client Application Components (Java/Apache CXF) appendix.

1.5.1.4.2 SOAP Message Security Header

For each client proxy you can define an interceptor with SOAP message security header content (for TeamConnect authentication). The interceptor should be added to the client proxy once and afterward, the corresponding security header will be automatically inserted into each XML message resulting from a Web Service request.

The interceptor for the message security header needs to contain the following:

- A string variable for the TeamConnect user name used to authenticate to the system.

  If you are writing a Web Service client application in a Software as a Service (SaaS) environment, the user name should use the format: user@domain

- A string variable for the TeamConnect user password used to authenticate to the system. During message transfer the password will be sent as unencrypted clear text.

  If you need to transmit the password in a more secure manner, you can set up SSL between your client application and the TeamConnect instance.

For a more detailed example, see the Security Headers Sample Code section of the Client Application Components (Java/Apache CXF) appendix.

The TeamConnect Web API is compliant with the WS-Security 1.1 OASIS standard. Currently only authentication using the UsernameToken is supported. For more information, see:

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

Sample SOAP Message Security Header for Authentication

In your custom application, you should define a security header for SOAP/HTTP requests that looks like:

```
<soap:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd" soap:mustUnderstand="1">
<wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/
01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
wsu:Id="UsernameToken-1809471" xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
<wsse:Username xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd">administrator</
wsse:Username>
<wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-
```

```
200401-wss-username-token-profile-1.0#PasswordText"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">password</wsse:Password>
</wsse:UsernameToken>
</wsse:Security>
</soap:Header>
//administrator should be replaced with a valid TeamConnect user name with
sufficient rights to perform required functions.
//password should be replaced with the administrator user's password.
```

1.5.1.4.3 Putting Components Together in a Client Application

In a client application there are two key items to do:

- Instantiate a client proxy and type cast the proxy with the Web Service client interface that corresponds to the entity type you pass as parameter. For example, if you use the entityName, Contact, then you'd cast the proxy as type ContactRepository.
- Add the interceptor for SOAP security header to the client proxy.

Afterward you would use the client proxy to make Web Service request calls.

For more a more detailed example, see the Client Application Sample Code section of the Client Application Components (Java/Apache CXF) appendix.

1.5.1.4.4 Common Functions

This section shows how you can use TeamConnect® Enterprise Web Services for common functions such as creating, updating, reading, searching, deleting records.

Each TeamConnect Web Service is represented by a repository WSDL file. Each Web Service corresponds to a TeamConnect entity or object type (for example, contacts, projects, accounts, invoices). There are a few recurring methods or operations that are common across all TeamConnect® Enterprise Web Services:

- Creating (Inserting) a record
- Updating a record
- Reading a record
- Searching for records
- Deleting a record

Some TeamConnect® Enterprise Web Services may have additional functions that are particular to that object, for example, posting or voiding records (such as invoices, expenses, tasks), and uploading documents, etc.

Note: *Some common functionality is not supported for involved party records.*

1.5.1.4.4.1 Creating Records

Each TeamConnect Web Service provides a method for creating a corresponding record. The general form of the method is insert*Object* (objectDataType). For example, for the Contact repository the corresponding method is insertContact (ContactCreate). The resulting record's unique key is returned.

1.5.1.4.4.2 Updating Records

Each TeamConnect Web Service provides a method for updating a corresponding record. The general form of the method is update*Object* (ObjectUpdate). For example, for the Contact repository the corresponding method is updateContact (ContactUpdate). Among the properties of the ContactUpdate object that you need to set, the unique key of the target record to update is required. There is no return value.

An update*Object* method does not retrieve all of a record's data before updating its properties. The target record to update is identified by its unique key, then you use set methods to update existing property values or populate empty property values.

To clear existing property values and reset the values to null, use the getClearedProps method provided in the *Object*Update class. For example, to update a contact of type person by clearing existing property values, use the PersonUpdate.getClearedProps().add(Item) method. Otherwise use the PersonUpdate.getClearedProps().addAll(Items) method where **Items** is a list of contact property names whose values to clear.

1.5.1.4.4.3 Reading Records

Each TeamConnect Web Service provides a method for reading a record of its object type. The general form of the method is read*Object* (String uniqueKey, List<String> properties). For example, for the Contact repository the corresponding method is readContact (String uniqueKey, List<String> properties). The corresponding contact record is returned with only the specified property values.

You can find the list of available properties in the types.wsdl file. For example, to find the properties available for contact records, look for the complex types: company, contact, or person. Each element for those types represents a property that can be returned on read. Similarly you can find the properties in the corresponding client-side source code (for example, in the Contact, Person, Company, ContAddress, ContEmailAddress objects, etc.). For more information, refer to Web Service API Reference.

1.5.1.4.4.4 Searching Records

Each TeamConnect Web Service provides a method for searching records by an object type and by additional search criteria. The general form of the method is read*Objects*ByCriteria (SearchCriteria criteria, int limit, List<String> properties). For example, for the Contact repository the corresponding method is readContactsByCriteria (SearchCriteria criteria, int limit, List<String> properties). A list of record objects that meet the search criteria is returned with only the specified property values.

Note: When defining the list of property values to return, remember that some of the entity properties are objects. In these cases you need to explicitly specify the property of the property (where the property type is object) to return. For an example, see Code Snippet for readContactsByCriteria (specify return properties of contact property of type object).

The search method provided in TeamConnect® Enterprise Web Services is designed to search among a moderate number of records (for example, if you expect less than 100 records in the search results). For the limit parameter, it is recommended to enter 100 as the maximum number of records to return. If you need to search across a larger volume of records, you may want to use the application UI.

A detailed description of how to define search criteria is provided below. The properties parameter in the readContactsByCriteria method is defined and used in the same way as for the readContact method (described above).

Search Criteria

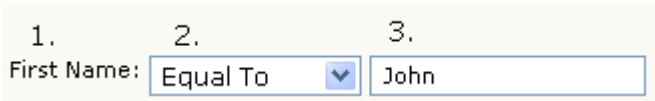
You need to define search criteria. From the \types directory you need to use the FieldSearchClause object to hold your search criteria. For each search, you need to define the following:

- one or more field criterion

  Field criterion are defined according to the data type for the field you are searching. For example, you can define field criterion for comparing a field that is a String type using the StringComparator and StringFieldCriterion objects.

- an operator (for example, AND or OR) that defines how the field criterion are combined in the search if you are using multiple field criteria

Each field criterion is made up of three parts: a field path (pointer to the database table/column that you will query), a comparator (defines how you are comparing the database record values against a literal value), and a literal value. The parts of a field criterion are illustrated in the following search fields screenshot.



Example of a Contact Search by First Name

In the contact search screenshot, First Name (1.) corresponds to the field path; Equal To (2.) corresponds to the comparator; and John (3.) is the literal value. Corresponding sample code for search criteria is in the Code Snippet section. See Searching for Contacts for a JAVA example of how to set search criteria.

Searching Records By Custom Field

When defining search criteria where you want to search for a record by a custom field value, you need to define the searchKeyPath using the following format: TCCategoryTreePosition + "__" + detailFieldName (where there are two underscores, "_", between the tree position and detail field name).

For example, fieldPathExpression.setSearchKeyPath("**DISP__CorrectiveActionDeadlineDl**");

For a more detailed example, see Code Snippet for readContactsByCriteria (search by custom field value).

Note: *If you will search by a custom field value that is associated with a child category, then the TCCategoryTreePosition should be constructed like parentObjectRootCategoryTreePosition + "_" + childCategoryTreePosition "__" + detailFieldName (where there is a single*

underscore, "_", between the root category tree position and its child category tree positions. Also note that an object's root category tree position is the same as the object definition's unique code.

1.5.1.4.4.5 Working with Categories

Web Services support enabling categories for a record, disabling categories for a record, and reading categories (and category properties) for a record.

Before You Begin

You need to get the following information before working with categories using Web Services:

- Category unique key—This value is constructed like <projectUniqueCode + "_" + categoryTreePosition>

  For example, if you are adding a parent category, Contracts, to a dispute record, then the category unique key would be "DISP_CONT"

  (where the tree position for Contracts is "CONT"; and the unique code for the Dispute object is "DISP")

  Another example would be if you are adding a child category, Confidentiality Breach, to a dispute record, then the category unique key would be "DISP_CONT_COBR"

  (where the following apply:

  - the tree position for the child category, Confidentiality Breach, is "COBR"
  - the tree position for the parent category, Contracts, is "CONT"
  - the unique code for the Dispute object is "DISP")
- Category name

Enabling Categories

When creating or updating a record, you can enable categories for the record. You can enable multiple categories for a record with the exception of invoice line items, tasks, and expenses.

Important: *A category must already exist in TeamConnect before you can enable the category for a record using Web Services. Web Services do not support creating a category for an object definition. Use the TeamConnect Designer UI area to create new categories.*

When creating records that are system object or custom object types, the TeamConnect system automatically sets the default category to the corresponding object definition's root category (for example an account record's root category is Account). As a result, you are not required to set a category for records with the exception of invoice line items.

When creating an invoice line item record where the line item type is set to EXPENSE, you need to explicitly set one Expense Category. Otherwise, when creating an invoice line item record where the line item type is set to FEE, you need to explicitly set one Fee/Task Category.

Caution: *If you set multiple categories for a line item, only the last set category will be saved.*

Note: *If a validation rule exists that requires new projects or matters to be saved with a category other than the root category, you may need to enable or add a category to the record when you create it using Web Services.*

From the Category type class, use the setUniqueKey (String) method to identify which category you want to add to (enable for) a record. From an *Object*Create class, you would use the getCategories() method to add an instance of the Category class for the type of record you are creating. For example:

```
Category cat = new Category();
cat.setUniqueKey("CONT_EXTE");
PersonCreate person = PersonCreate();
person.getCategories().add(cat);
```

Use the getCategories() method to:

- enable categories for a record during record creation or
- enable additional categories for a record during record update

Reading Primary Categories

If a record has multiple categories and you are only interested in getting the primary category, include primaryCategory in the properties parameters.

Disabling Categories

When updating a record, use the getCategoryDeleteUniqueKeys() method to disable a category for that record. For example:

```
PersonUpdate person = PersonUpdate();
person.getCategoryDeleteUniqueKeys().add("CONT_INTR");
\\where "CONT_INTR" is the unique key of the category to disable
```

Reading Categories

Reading categories for a record is different from reading other record properties in that in the properties (List of string property names) parameter, you include "categories". The returned record object will include a list of category objects for that record (including all of that record's custom fields).

Important: *Reading categories is the exception to the general rule that when you specify a record's properties to read, for properties that are objects, you need to specify the non-object property of the parent object property to retrieve.*

For example:

```
Person readPerson = (Person)contactRepository.readContact(uniqueKey,
getPropertiesToRead());
}
```

```
private List<String> getPropertiesToRead() {
    List<String> properties = new ArrayList<String>();
    properties.add("firstName");
    properties.add("lastName");
    properties.add("address.type");
    properties.add("categories");
    return properties;
}
```

1.5.1.4.4.6 Working with Custom Fields

Web Services support populating, editing, clearing, and reading custom field values.

Before You Begin

A custom field must already exist in TeamConnect before you can add the custom field to a record using Web Services. Use the TeamConnect Designer UI area to create new custom fields (for an object/category combination).

Important: *Each custom field is dependent on a record category. Before you can work with a custom field in a record, its corresponding category (or category hierarchy) must be enabled in that record. For more information, see Working with Categories.*

You also need to get the following information before working with categories using Web Services:

- Category unique key—This value is constructed like <projectUniqueCode + "_" + categoryTreePosition>

  For example, if you are adding a parent category, Contracts (with tree position "CONT") that was created under the Dispute object (with unique code "DISP"), then the category unique key would be "DISP_CONT".

  Another example would be if you are adding a child category, Confidentiality Breach (with tree position "COBR"), which is a child of the Contracts category under the Dispute object, then the unique key would be "DISP_CONT_COBR".

- Category name
- Custom field type (for example, Boolean, DateTime, Decimal, Involved, Note, Project, Text, TableItem)
- Custom field name

Populating and Updating Custom Field Values

To populate a custom field value, you would enable the corresponding category for a record. There are several type classes available to instantiate a class or container for the properties of a custom field. Each of these type classes are associated with a field data type and include: BooleanCustomField, DateTimeCustomField, DecimalCustomField, InvolvedCustomField, NoteCustomField, ProjectCustomField, TableItemCustomField, TextCustomField. Instantiate the appropriate custom field type class. Use the provided methods, setFieldName(String value) and

setValue(boolean value) to identify the custom field name to populate and to assign the field value for, respectively (where the setValue parameter type would vary).

Each custom field is associated with a category. From the Category type class, several methods are provided to add the custom field data object described in the paragraph above, including getBooleanCustomFields(); getDateTimeCustomFields(); getDecimalCustomFields(); getInvolvedCustomFields(); getNoteCustomFields(); getProjectCustomFields(); getTableItemCustomFields(); getTextCustomFields().

A sample for populating a boolean custom field (associated with a corresponding category) for a record follows:

```
PersonCreate person = createPerson();
Category cat2 = new Category();
cat2.setUniqueKey("CONT");
BooleanCustomField synched = new BooleanCustomField();
synched.setFieldName("Synchronized");
synched.setValue(true);
cat2.getBooleanCustomFields().add(synched);
person.getCategories().add(cat2);
```

Use the custom field type class' setValue() method to populate or update a custom field's value.

Clearing Custom Field Values

Use the custom field type class' setValue() method to clear a custom field's value. For example:

```
PersonUpdate person = PersonUpdate();
Category cat2 = new Category();
cat2.setUniqueKey("CONT");
BooleanCustomField synched = new BooleanCustomField();
synched.setFieldName("Synchronized");
synched.setValue();
//in this case set the value to null to clear the current custom field value.
if the custom field were a String type, you'd set the value to ""
cat2.getBooleanCustomFields().add(synched);
person.getCategories().add(cat2);
```

Reading Custom Field Values

When reading a record, specify the categories property as part of the return values. The resulting record's categories (property) object will also contain all associated custom fields and related custom field properties.

1.5.1.4.4.7 Deleting Records

Each TeamConnect Web Service provides a method for deleting a corresponding record. The general form of the method is delete*Object* (String uniqueKey). For example, for the Contact repository the corresponding method is deleteContact (String uniqueKey). There is no return value.

1.5.2 Web Service API Reference

This chapter provides a reference for the TeamConnect Web Service repositories, their request methods, parameters, and return values. Summary information is also provided for the data objects or record property containers typically specified as request parameters.

Data Object Summary Tables

This guide does not include detailed property descriptions for data objects (containers for record property values). Most of these properties and their data restrictions are documented in the Customization guide, Object Model Reference section. In the API Reference Summary section of this chapter, table summaries listing the data objects typically used within a repository's requests or returned from the server are provided. Those table summaries include references to the corresponding Object Model Reference object names.

To locate more detailed descriptions of a TeamConnect Web Service data object's property, find the corresponding Object Model Reference object name. Search for that Object Model Reference object name in the Customization guide or TeamConnect Designer area help. From the description of each Object Model Reference object, you can typically match the TeamConnect Web Service data object property name directly to the attribute name for the Object Model Reference object (although exceptions may exist).

Note: *For example, if you were interested in learning more information about the property,* ***name****, for the Web Service data object, account, then you would search the Customization help/guide for the Object Model Reference attribute table, "TAccount" and within the resulting help/guide section, look for the Attribute,* ***name****, and its related Comments.*

There are a set of data objects used to set search criteria. Each of these data objects can generally be used to search multiple record types so they are listed separately from the repository summary tables. See Data Objects Used in Search Criteria for more information.

System Lookup Table Items

There are a set of data objects that you use to populate a record (repository) property value(s) using existing items from TeamConnect system lookup tables. These data objects (classes) are subclasses of the lookupItem abstract class.

To add a system lookup table item to populate a record property, use the method, setStoredValue (String storedValue), where the storedValue should be the TeamConnect system lookup table item's Tree Position value. Note that the system lookup table item should already be defined in TeamConnect before you can use the item with Web Services.

For an example, Code Snippet for creating a contact of type person and populating a property from a system lookup table.

1.5.2.1 API Reference Summary

This section provides the following information per repository (Web Service interface):

- A table of requests
- A table of required fields to create a record with the repository

- A table of related data objects (record property containers)

Note: *There are some remaining data objects (for example, related to search criteria), that are currently not documented in this section.*

1.5.2.1.1 AccountRepository

AccountRepository Requests Summary

Request	Description
activateAccount	Activates an account.
allocateMoney	Deposits funds to an account.
deactivateAccount	Deactivates an account.
deleteAccount	Deletes an account record.
insertAccount	Creates an account record.
readAccount	Reads and returns an account record. Specified account properties are returned with the record.
readAccountsByCriteria	Searches for account records that meet a given search criteria. Specified account properties are returned with the resulting records.
readChildAccounts	Reads child account records of a given parent account.
readRecentlyViewedAccounts	Reads and returns recently viewed account records.
transferMoney	Transfers funds between accounts.
updateAccount	Updates an existing account with the specified properties.
withdrawMoney	Withdraws funds from an account.

AccountRepository Data Objects Summary

Data Object	Description

1 account 2 accountCreate 3 accountUpdate	1 Account record property container returned when you read or search accounts. 2 Account record property container used to set a new account's properties. This object is passed as a request parameter when you create (insert) an account. 3 Account record property container used to update or clear an account's properties. This object is passed as a request parameter when you update an account. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the TAccount object.*
accountType	An enumeration used to set the account type. Options include: • BUDGET • RESERVE
accountOverdraftType	An enumeration used to set the account overdraft type. Options include: • NON_NEGATIVE • ALLOW_NEGATIVE • OVERDRAFT
accountInvolvedType	An enumeration used to set restrictions on the posting criteria for an account. The effect of this enumeration's value depends on other account posting criteria you have set in either the accountCreate or accountUpdate properties. If posting criteria is applied to tasks or expenses, use this enumeration to restrict account posting by the task/expense's associated contact. If posting criteria is applied to invoice line items, use this enumeration to restrict account posting by the line item's associated timekeeper. Options include: • ANY_INVOLVED • ONE_INVOLVED
accountProjectType	An enumeration used to set restrictions on the posting criteria for an account. The effect of this enumeration's value depends on other account posting criteria you have set in either the accountCreate or accountUpdate properties.

	Use this enumeration to restrict account posting by the task/ expense/invoice line item's associated project. Options include: • ANY_PROJECT • ONE_PROJECT • BY_TYPE
accountVendorType	An enumeration used to set restrictions on the posting criteria for an account. The effect of this enumeration's value depends on other account posting criteria you have set in either the accountCreate or accountUpdate properties. Use this enumeration to restrict account posting by the invoice line item's associated vendor. Options include: • ANY_VENDOR • ONE_VENDOR

1.5.2.1.2 AppointmentRepository

AppointmentRepository Requests Summary

Request	Description
insertAppointment	Saves a new appointment to the repository.
updateAppointment	Updates a previously saved appointment.
readAppointment	Retrieves a appointment from the repository based on a unique key. For a list of available properties, see the type class, appointment.
readAppointmentsByCriteria	Retrieves a set of appointments based on specified criteria. If the criteria is empty, then the search will be for all appointments. The specified limit takes precedence over the current system setting for the maximum number of search results to obtain. For a list of available properties, see the type class, appointment.
deleteAppointment	Deletes the appointment specified by the unique key.

readRecentlyViewedAppointments	Read and return recently reviewed appointments.

AppointmentRepository Data Objects Summary

Data Object	Description
1 appointment 2 appointmentCreate 3 appointmentUpdate	1 Appointment record property container returned when you read or search appointments. When defining a repository search's return properties, refer to the property names defined in this object. 2 Appointment record property container used to set a new appointment's properties. This object is passed as a request parameter when you create (insert) an appointment. 3 Appointment record property container used to update or clear an appointment's properties. This object is passed as a request parameter when you update an appointment. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the TAppointment object.*
1 appointmentResource 2 appointmentResourceCreate 3 appointmentResourceUpdate	1 Appointment resource object property container returned when you read or search appointment resources. 2 Appointment resource object property container used to set a new appointment resource's properties. 3 Appointment resource object property container used to update or clear an appointment resource's properties.
appointmentResourceType	Subclass of lookupItem. Use the provided get methods to read an appointment resource's associated type.
lookupItem	Abstract class that provides get methods to read properties associated with TeamConnect system lookup table items.
1 attendee 2 attendeeCreate 3 attendeeUpdate	1 Attendee object property container returned when you read or search attendees. 2 Attendee object property container used to set a new attendee's properties. 3 Attendee object property container used to update or clear an attendee's properties.

attendanceType	An enumeration used to indicate an attendee's plans to attend an appointment. Options include: • WILL_ATTEND • TENTATIVE • WILL_NOT_ATTEND • UNKNOWN

1.5.2.1.3 ContactRepository

ContactRepository Requests Summary

Request	Description
deleteContact	Deletes the contact specified by the unique key.
insertContact	Saves a new Contact object to the repository.
readContact	Retrieves a Contact from the repository based on a unique key. For a list of available properties, see the type class, contact.
readContactsByCriteria	Retrieves a set of contacts based on specified criteria. If the criteria is empty, then the search will be for all contacts. The specified limit takes precedence over the current system setting for the maximum number of search results to obtain. For a list of available properties, see the type class, contact.
ReadRecentlyViewedContacts	Retrieve a list of recently viewed contacts.
updateContact	Updates a previously saved contact.

ContactRepository Data Objects Summary

Data Object	Description
1 Company 2 CompanyCreate 3 CompanyUpdate	All three objects are subclasses of the related Contact objects. The Company objects are used to work with contact records of type, Company. 1 Company contact record property container returned when you read or search companies. When defining a repository search's

	return properties, refer to the property names defined in this object. 2 Company contact record property container used to set a new company's properties. This object is passed as a request parameter when you create (insert) a company. 3 Company contact record property container used to update or clear a company's properties. This object is passed as a request parameter when you update a company. ***Note:*** *CompanyCreate is a subclass of ContactCreate. CompanyUpdate is a subclass of ContactUpdate.* ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the TContact object.*
1 Person 2 PersonCreate 3 PersonUpdate	All three objects are subclasses of the related Contact objects. The Person objects are used to work with contact records of type, Person. 1 Person contact record property container returned when you read or search person contacts. When defining a repository search's return properties, refer to the property names defined in this object. 2 Person contact record property container used to set a new person contact's properties. This object is passed as a request parameter when you create (insert) a person contact. 3 Person contact record property container used to update or clear a person contact's properties. This object is passed as a request parameter when you update a person contact. ***Note:*** *PersonCreate is a subclass of ContactCreate. PersonUpdate is a subclass of ContactUpdate.* ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the TContact object.*
1 ContAddress 2 ContAddressCreate 3 ContAddressUpdate	All three objects are used to work with contact mailing addresses within contact records. 1 Contact address object property container returned when you read or search contacts, including the ContAddress property.

	2 Contact address object property container used to set properties for a contact address. 3 Contact address object property container used to update or clear properties for a contact address. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JContAddress object.*
contAddressType	Subclass of lookupItem. Use the provided get methods to read the properties related to a contact address type.
countryItem	Subclass of lookupItem. Use the provided get methods to read the properties related to a contact address country.
1 ContEmailAddress 2 ContEmailAddressCreate 3 ContEmailAddressUpdate	All three objects are used to work with contact email addresses within contact records. 1 Contact email address object property container returned when you read or search contact email addresses. 2 Contact email address object property container used to set properties for a contact email address. 3 Contact email address object property container used to update or clear properties for a contact email address. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JContEmail object.*
contEmailAddressType	Subclass of lookupItem. Use the provided get methods to read the properties related to a contact email address' associated type.
1 ContFaxNumber 2 ContFaxNumberCreate 3 ContFaxNumberUpdate	All three objects are used to work with contact fax numbers within contact records. 1 Contact fax number object property container returned when you read or search contact fax numbers. 2 Contact fax number object property container used to set properties for a new contact fax number. 3 Contact fax number object property container used to update or clear properties for a contact fax number.

	Note: *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JContFax object.*
contFaxNumberType	Subclass of lookupItem. Use the provided get methods to read the properties related to a contact fax number's associated type.
1 ContInetAddress 2 ContInetAddressCreate 3 ContInetAddressUpdate	All three objects are used to work with contact internet addresses (website URL's) within contact records. 1 Contact internet address object property container returned when you read or search contact internet addresses. 2 Contact internet address object property container used to set properties for a new contact internet address. 3 Contact internet address object property container used to update or clear properties for a contact internet address. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JContInetAddress object.*
contInetAddressType	Subclass of lookupItem. Use the provided get methods to read the properties related to a contact email address' associated type.
1 ContPhoneNumber 2 ContPhoneNumberCreate 3 ContPhoneNumberUpdate	All three objects are used to work with contact phone numbers within contact records. 1 Contact phone number object property container returned when you read or search contact phone numbers. 2 Contact phone number object property container used to set properties for a new contact phone number. 3 Contact phone number object property container used to update or clear properties for a contact phone number. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JContPhone object.*
contPhoneNumberType	Subclass of lookupItem. Use the provided get methods to read the properties related to a contact phone number's associated type.
1 ContDefaultRate 2 ContDefaultRateCreate	All three objects are used to work with contact default rates within contact records.

3 ContDefaultRateUpdate	1 Contact default rate object property container returned when you read or search contact default rates. 2 Contact default rate object property container used to set properties for a new contact default rate. 3 Contact default rate object property container used to update or clear properties for a contact default rate. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JContRate object.*
1 contTaskRate 2 contTaskRateCreate 3 contTaskRateUpdate	All three objects are used to work with contact task rates within contact records. 1 Contact task rate object property container returned when you read or search contacts. 2 Container used to set properties for contact invoice task rates. 3 Contact rate object property container used to update or clear properties for contact invoice task rates. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JContRate object.*
1 contInvoiceTaskRate 2 contInvoiceTaskRateCreate 3 contInvoiceTaskRateUpdate	All three objects are used to work with contact invoice task rates within contact records. 1 Contact invoice task rate object property container returned when you read or search contacts. 2 Container used to set properties for contact invoice task rates. 3 Contact rate object property container used to update or clear properties for contact invoice task rates. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JContRate object.*
1 contDefaultRate 2 contDefaultRateCreate 3 contDefaultRateUpdate	All three objects are used to work with default contact rates within contact records. 1 Default contact rate object property container returned when you read or search contacts. 2 Default contact rate object property container used to set properties for a default contact rate.

	3 Default contact rate object property container used to update or clear properties for a default contact rate. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JContRate object.*
currencyItem	Subclass of lookupItem. This class corresponds to the TeamConnect Multi-currency lookup table's items. Use the provided get methods to read the properties related to either a contact rate's currency or invoice currency. The currencyItem name property corresponds to the TeamConnect multi-currency lookup table Currency code. The currencyItem name property
1 ContRelation 2 ContRelationCreate 3 ContRelationUpdate	All three objects are used to work with contact relations within contact records. 1 Contact relation object property container returned when you read or search contact relations. 2 Contact relation object property container used to set properties for a new contact relation. 3 Contact relation object property container used to update or clear properties for a contact relation. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JContRelation object.*
contRelationType	Subclass of lookupItem. Use the provided get methods to read the properties related to a contact relation's associated type.
relationDirection	An enumeration used to identify the direction between two relations. Options include: • FROM_THIS • TO_THIS
1 ContSkill 2 ContSkillCreate 3 ContSkillUpdate	All three objects are used to work with contact skills within contact records. 1 Contact skill object property container returned when you read or search contact skills.

	2 Contact skill object property container used to set properties for a new contact skill. 3 Contact skill object property container used to update or clear properties for a contact skill. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JContSkill object.*
contSkillType	Subclass of lookupItem. Use the provided get methods to read the properties related to a contact skill's associated type.
1 ContTerritory 2 ContTerritoryCreate 3 ContTerritoryUpdate	All three objects are used to work with contact territories within contact records. 1 Contact territory object property container returned when you read or search contact territories. 2 Contact territory object property container used to set properties for a new contact territory. 3 Contact territory object property container used to update or clear properties for a contact territory. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JContTerritory object.*
contTerritoryItem	Subclass of lookupItem. Use the provided get methods to read the properties related to a contact territory.

1.5.2.1.4 DocumentRepository

DocumentRepository Requests Summary

Request	Description
insertDocument	Saves a new Document object to the repository.
createShortcut	Creates a shortcut or link to a document (file, folder, or existing shortcut) with a given unique key. The parentFolderUniqueKey defines the folder where the shortcut will be created.

createSubFolder	Creates a sub-folder with given name in given parent folder.
createHyperlinkDefaultCategory	Creates a hyperlink and sets its category to the default Document category.
createHyperlink	Creates a hyperlink.
updateDocument	Updates a previously saved document.
readDocument	Retrieves a Document from the repository based on a unique key. For a list of available properties, see the type class, Document.
readDocumentsByCriteria	Retrieves a set of documents based on specified criteria. If the criteria is empty, then the search will be for all documents. The specified limit takes precedence over the current system setting for the maximum number of search results to obtain. For a list of available properties, see the type class, Document.
readChildDocuments	Performs two functions: • For a given document file (parent), gets older document versions of that file. • For a given folder, gets documents within that folder. Returns documents with only specified property values. For a list of available properties, see the type class, document.
readChildDocumentForName	Retrieves a document from the repository given the file name and the unique ID of its parent folder.
readDocumentFolderForDocumentOwner	Retrieves a document folder from the repository given the unique key of the folder's owner (user).
readDocumentByPath	Retrieves a Document from the repository based on a folder path.
checkOut	Attempts to check out a document with a given unique key.
undoCheckOut	Attempts to undo check out of a given document.

checkIn	Attempts to check in a document with a given unique key.
revert	Attempts to revert a given document to a given old version.
moveDocument	Moves a document to a new parent folder.
copyDocument	Copies a document to a new parent folder.
deleteDocument	Deletes the persistent document specified by the unique key.
ReadRecentlyViewedDocuments	Retrieves a list of recently viewed documents.
setDocumentAsRecentlyViewed	Set a document as recently viewed.
readFolderHierarchy	Retrieves a hierarchical list of document folders for one or more projects.

DocumentRepository Data Objects Summary

Data Object	Description
1 document 2 documentCreate 3 documentUpdate	1 Document record property container returned when you read or search documents. When defining a repository search's return properties, refer to the property names defined in this object. 2 Document record property container used to set a new document's properties. This object is passed as a request parameter when you create (insert) a document. 3 Document record property container used to update or clear a document's properties. This object is passed as a request parameter when you update a document. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the TDocument object.*
documentContentType	This object defines the document file type. In an application you would typically use this object's properties to read or search documents. ***Note:*** *For more information about the data restrictions for properties that belong to this object, see the Object Model Reference for the YDocuContentType object.*

documentType	An enumeration used to indicate the document type. Options include: • FILE • FOLDER • HYPERLINK • SHORTCUT • HISTORY—the documentType for previous versions of a document file. Note that a document with documentType HISTORY is different from a history record. ***Note:*** *For more information about the data restrictions for properties that belong to this object, see the Object Model Reference for the YDocuContentType object.*

1.5.2.1.5 ExpenseRepository

ExpenseRepository Requests Summary

Request	Description
deleteExpense	Deletes an expense record.
insertExpense	Creates an expense record.
postExpense	Posts an expense record.
readExpense	Reads and returns an expense record. Specified expense properties are returned with the record.
readExpensesByCriteria	Searches for expense records that meet a given search criteria. Specified invoice properties are returned with the resulting records.
readRecentlyViewedExpenses	Read and return recently viewed expense records.
updateExpense	Updates an expense record. Specified invoice properties are updated.
voidExpense	Voids an expense record.

ExpenseRepository Data Objects Summary

Data Object	Description
accountPostingStatus	An enumeration used to track whether a finance record has posted to an account. Typically in an application, you would read or search by this object's value. You would not explicitly set a financial record's posting status. Options include: • NOT_SUBMITTED • POSTED • FAILED
1 expense 2 expenseCreate 3 expenseUpdate	1 Expense record property container returned when you read or search expenses. 2 Expense record property container used to set a new expense's properties. This object is passed as a request parameter when you create (insert) an account. 3 Expense record property container used to update or clear an expense's properties. This object is passed as a request parameter when you update an expense. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the TExpense object.*

1.5.2.1.6 GroupAccountRepository

GroupAccountRepository Requests Summary

Request	Description
deleteGroupAccount	Deletes the group account specified by the unique key.
insertGroupAccount	Saves a new group account object to the repository.
readGroupAccount	Retrieves a group account from the repository based on a unique key. For a list of available properties, see the type class, groupAccount.
readGroupAccountsByCriteria	Retrieves a set of group accounts based on specified criteria. If the criteria is empty, then the search will be for all group accounts. The specified limit takes precedence over the current system setting for the maximum number of

	search results to obtain. For a list of available properties, see the type class, groupAccount.
readRecentlyViewedGroupAccounts	Read and return recently viewed group accounts.
updateGroupAccount	Updates a previously saved group account.

GroupAccountRepository Data Objects Summary

Data Object	Description
1 groupAccount 2 groupAccountCreate 3 groupAccountUpdate	1 Group account record property container returned when you read or search groups. 2 Group account record property container used to set a new group's properties. This object is passed as a request parameter when you create (insert) a group. 3 Group account record property container used to update or clear a group's properties. This object is passed as a request parameter when you update a group. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the YGroup object.*

1.5.2.1.7 HistoryRepository

HistoryRepository Requests Summary

Request	Description
insertHistory	Saves a new history record to the repository.
readHistoriesByCriteria	Retrieves a set of records based on specified criteria. If the criteria is empty, then the search will be for all records. The specified limit takes precedence over the current system setting for the maximum number of search results to obtain. For a list of available properties, see the type class, history.
readHistory	Retrieves a record from the repository based on a unique key. For a list of available properties, see the type class, history.
updateHistory	Updates a previously saved history record.

deleteHistory	Deletes the record specified by the unique key.
readRecentlyViewedHistories	Read recently viewed histories.

HistoryRepository Data Objects Summary

Data Object	Description
1 history 2 historyCreate 3 historyUpdate	1 History record property container returned when you read or search history records. When defining a repository search's return properties, refer to the property names defined in this object. 2 History record property container used to set a new history record's properties. This object is passed as a request parameter when you create (insert) a history record. 3 History record property container used to update or clear a history record's properties. This object is passed as a request parameter when you update a history record. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the THistory object.* ***Note:*** *historyOwner refers to the unique key (String) of a parent record that the history describes.*

1.5.2.1.8 InvoiceRepository

InvoiceRepository Requests Summary

Request	Description
adjustInvoiceHeader	Adjust an invoice on the header level.
deleteInvoice	Deletes an invoice record.
insertInvoice	Creates an invoice record.
postInvoice	Posts an invoice record.
readActiveApprovals	Read active approvals for an invoice.
readCompletedApprovals	Read completed approvals for an invoice.

readInvoice	Reads and returns an invoice record. Specified invoice properties are returned with the record.
readInvoiceApprovalsPendingOnPost	Read all pending invoice approvals that are pending on post.
readInvoicesByCriteria	Searches for invoice records that meet a given search criteria. Specified invoice properties are returned with the resulting records.
readRecentlyViewedInvoices	Read recently viewed invoices.
updateInvoice	Updates an invoice record. Specified invoice properties are updated.
voidInvoice	Voids an invoice record.

InvoiceRepository Data Objects Summary

Data Object	Description
adjustmentMethod	An enumeration used to identify the way an invoice summary is adjusted. Options include: • REDUCE_BY_AMOUNT • REDUCE_BY_PERCENT • NEW_AMOUNT ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JInvcHeaderAdjustment object.*
currencyItem	See currencyItem.
1 invoiceAdjustment 2 invoiceAdjustmentCreate 3 invoiceAdjustmentUpdate	All three objects are used to work with invoice summary adjustments within invoice records. 1 Invoice summary adjustment object property container returned when you read or search invoices. 2 Invoice summary adjustment object property container used to set properties for an invoice. 3 Invoice summary adjustment object property container used to update an invoice.

	Note: *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JInvcHeaderAdjustment object.*
invoiceAdjustmentTarget	An enumeration used to identify which property of the invoice summary to adjust. Options include: • TOTAL_FEES • TOTAL_EXPENSES • TOTAL_INVOICE
1 invoice 2 invoiceCreate 3 invoiceUpdate	1 Invoice record property container returned when you read or search invoices. 2 Invoice record property container used to set a new invoice's properties. This object is passed as a request parameter when you create (insert) an invoice. 3 Invoice record property container used to update or clear an invoice's properties. This object is passed as a request parameter when you update an invoice. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the TInvoice object.*
invoiceNonUSTax	Invoice Non-US Taxes object property container returned when you read or search invoices. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned object, see the Object Model Reference for the TInvoice object.*
invoiceNonUSTaxType	Subclass of lookupItem. Use the provided get methods to read the properties related to an invoice's Non-US Tax category. ***Note:*** *An invoice can have multiple Non-US Tax amounts.*
invoicePostingStatus	An enumeration used to track whether a finance record has posted to an account. Typically in an application, you would read or search by this object's value. You would not explicitly set a invoice's posting status. Options include:

	• NOT_SUBMITTED • POSTED • FAILED • REJECTED
1 lineItemAdjustment 2 lineItemAdjustmentCreate 3 lineItemAdjustmentUpdate	All three objects are used to work with invoice line item adjustments within invoice records. 1 Line item adjustment object property container returned when you read or search invoices. 2 Line item adjustment object property container used to set properties for an invoice line item adjustment. 3 Line item adjustment object property container set properties for an invoice line item adjustment. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JInvcLineItem object.*
lineItemAdjustmentReason	Subclass of lookupItem. Use the provided get methods to read the properties related to a line item adjustment reason. This class corresponds to the options defined by the TeamConnect system lookup table, Invoice Rejection Reason.
lineItemAdjustmentTarget	An enumeration that identifies the line item property to adjust. Options include: • RATE • QUANTITY • DISCOUNT • TOTAL
1 lineItem 2 lineItemCreate 3 lineItemUpdate	1 Line Item record property container returned when you read or search invoices. 2 Line Item record property container used to create a line item and add it to an invoice. 3 Line Item record property container used to update a line item. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JInvcLineItem object.*

lineItemType	An enumeration that sets the line item type. Options include: • EXPENSE • FEE
taskActivityItem	Subclass of lookupItem. Use the provided get methods to read the properties related to a invoice line item task activity option. This class corresponds to options defined in the TeamConnect system lookup table, Activity Item.

1.5.2.1.9 LookupTableSource

LookupTableSource Requests Summary

Request	Description
readSystemLookupTable	Retrieves a list of properties for a system table code. You use the properties to read values from lookup tables.

1.5.2.1.10 ProjectRepository

Use the ProjectRepository to manage project, matters, or custom object records (for example, create, update, read,search, delete). The corresponding custom object must already be defined in the TeamConnect Designer area Object Definitions screen. Corresponding categories for the custom object must also be defined in the TeamConnect Designer area Object Definitions screen.

ProjectRepository Requests Summary

Request	Description
changePhase	Changes the phase of a project record. The phase must already be defined in TeamConnect.
deleteProject	Deletes a project record.
insertProject	Creates a project record.
readChildProjectsForEntityType	Reads and returns a project record's child projects by a given category. Specified project properties are returned with the resulting records.
readProject	Reads and returns a project record. Specified project properties are returned with the record.
readProjectEntityTypes	Get a list of project entity types in TeamConnect.

readProjectIntegrationSearches	Get a list of search views that are of type "Integration", each returned search view should have a unique key, a name, and its associated entity type.
readProjectsByCriteria	Searches for project records that meet a given search criteria. Specified project properties are returned with the resulting records.
readProjectsUsingSearch	Get a list of records that match the search criteria given a unique key of search view of "Integration" type.
readRecentlyViewedProjects	Read recently viewed projects.
updateProject	Updates a project record. Specified project properties are updated.

ProjectRepository Data Objects Summary

Data Object	Description
assigneeRole	Use this object to read or search projects by associated assignee role. ***Note:*** *Project assignee role values must be pre-defined in TeamConnect UI in the Project (custom object) object definition.*
1 embeddedEntity 2 embeddedEntityCreate 3 embeddedEntityUpdate	1 Embedded object record property container returned when you read or search projects. 2 Embedded object record property container used to set properties of an embedded object record. 3 Embedded object record property container used to set properties of an embedded object record. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the WObjdProjectInfo object.*
embeddedEntityType	Use this object's uniqueCode property to identify the existing embedded object definition to work with.
phase	Use this object's properties to search or read projects. Also use this object's properties to search or read projects by the **phaseInterval** (object) property.

phaseInterval	The phaseInterval object is a container for properties about a phase's duration (for example, the date when a phase became active). In an application, typically you would Project phase interval object property container returned when you read or search projects. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned object, see the Object Model Reference for the JProjPhase object.*
1 projectAssignee 2 projectAssigneeCreate 3 projectAssigneeUpdate	1 Project assignee record property container returned when you read or search projects. 2 Project assignee record property container used to set a project's properties. 3 Project assignee record property container used to set a project's properties. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JProjAssignee object.*
1 project 2 projectCreate 3 projectUpdate	1 Project (custom object) record property container returned when you read or search projects. 2 Project (custom object) record property container used to set a new project's properties. This object is passed as a request parameter when you create (insert) a project. 3 Project (custom object) record property container used to update or clear a project's properties. This object is passed as a request parameter when you update a project. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the TProject object.*
1 projRelation 2 projRelationCreate 3 projRelationUpdate	1 Project relation record property container returned when you read or search projects. 2 Project relation record property container used to set a project's properties. 3 Project relation record property container used to set a project's properties.

	Note: *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JProjRelation object.*
projRelationType	Subclass of lookupItem. Use the provided get methods to read the properties related to a project relation's associated type.
relationDirection	For more information, see relationDirection.

1.5.2.1.11 InvolvedRepository

Use this repository to work with involved party records or contacts who are involved in a project (matter).

InvolvedRepository Requests Summary

Request	Description
deleteInvolved	Deletes an involved party record.
insertInvolved	Inserts or creates an involved party record (associated with a project/ matter).
readInvolved	Reads and returns an involved record. Specified involved properties are returned with the record.
readInvolvedsByCriteria	Searches for involved party records that meet a given search criteria. Specified involved party properties are returned with the resulting records.
readInvolvedsForProject	Reads and returns a list of involved party records given the unique id of an associated project (matter). Specified involved properties are returned with the record.
updateInvolved	Updates an involved record. Properties that can be updated include categories and custom field values. Existing property values can be cleared.

InvolvedRepository Data Objects Summary

Data Object	Description
1 involved 2 involvedCreate	1 Involved record property container provides properties returned when you read or search involved parties. When defining a repository search's

3 involvedUpdate	return properties, refer to the property names defined in this object. 2 Involved record property container used to set a new involved party's properties. 3 Involved record object property container used to update or clear an involved party's properties. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the TInvolved object.*

1.5.2.1.12 TaskRepository

TaskRepository Requests Summary

Request	Description
deleteTask	Deletes a task record.
insertTask	Creates a task record.
postTask	Posts a task record.
readRecentlyViewedTasks	Read recently viewed tasks.
readTask	Reads and returns a task record. Specified task properties are returned with the record.
readTasksByCriteria	Searches for task records that meet a given search criteria. Specified task properties are returned with the resulting records.
reassign	Reassigns a task record to a different user.
updateTask	Updates a task record. Specified task properties are updated.
voidTask	Voids a task record.

TaskRepository Data Objects Summary

Data Object	Description
accountPostingStatus	See accountPostingStatus.

1 taskAssignee 2 taskAssigneeCreate 3 taskAssigneeUpdate	1 Task assignee record property container provides properties returned when you read or search tasks. When defining a repository search's return properties, refer to the property names defined in this object. 2 Task assignee record property container used to set properties for a Task assignee. 3 Task assignee record object property container used to update or clear properties for a Task assignee. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the JTaskAssignee object.*
taskPriority	An enumeration used to indicate the priority of a task. Options include: • HIGHEST • HIGH • NORMAL • LOW • LOWEST
1 task 2 taskCreate 3 taskUpdate	1 Task record property container returned when you read or search tasks. 2 Task record property container used to set a new task's properties. This object is passed as a request parameter when you create (insert) a task. 3 Task record property container used to update or clear a task's properties. This object is passed as a request parameter when you update a task. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the TTask object.*
taskWorkStatus	An enumeration used to indicate the status of a task (like level of completion). Options include: • NOT_STARTED • STARTED • COMPLETED

1.5.2.1.13 UserAccountRepository

UserAccountRepository Requests Summary

Request	Description
insertUserAccount	Saves a new user account object to the repository.
updateUserAccount	Updates a previously saved user account.
readUserAccount	Retrieves a user account from the repository based on a unique key. For a list of available properties, see the type class, userAccount.
readUserAccountsByCriteria	Retrieves a set of user accounts based on specified criteria. If the criteria is empty, then the search will be for all user accounts. The specified limit takes precedence over the current system setting for the maximum number of search results to obtain. For a list of available properties, see the type class, userAccount.
deleteUserAccount	Deletes the user account specified by the unique key.
readRecentlyViewedUserAccounts	Read recently viewed user accounts.

UserAccountRepository Data Objects Summary

Data Object	Description
1 userAccount 2 userAccountCreate 3 userAccountUpdate	1 User account record property container returned when you read or search users. 2 User account record property container used to set a new user's properties. This object is passed as a request parameter when you create (insert) a user. 3 User account record property container used to update or clear a user's properties. This object is passed as a request parameter when you update a user. ***Note:*** *For more information about the data restrictions for properties that belong to the above-mentioned objects, see the Object Model Reference for the YUser object.* ***Note:*** *To update a user's password through Web Services, you need to authenticate with that user name and password.*

userType	An enumeration used to indicate a user's access to public and private records. Options include: • SUPER • NORMAL • LIMITED

1.5.2.2 Repository Method Details

This section provides more detailed information about the methods or requests supported for repositories.

1.5.2.2.1 AccountRepository Method Details

This section describes the AccountRepository methods, including parameters and return values. For the method to insert a new record, a list of required properties is provided.

1.5.2.2.1.1 insertAccount

Inserts or creates an account record.

Parameters

insertAccount Parameter Descriptions

Parameter Name	Data Type	Description
accountCreate	accountCreate	Data object

Required Properties from accountCreate

The following table lists properties from the accountCreate data object that must be populated in order to create an account. Also note that the same properties must be populated when you are updating an account, using the accountUpdate data object.

accountCreate Required Properties

Property Name	Data Type	Description
name	string	The account name.
allocationLimit	decimal	The maximum amount that can be allocated to the account.
startOn	dateTime	The account start date.

endOn	dateTime	The account end date.

Response

insertAccount Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the newly created account record.

1.5.2.2.1.2 updateAccount

Updates an account record.

When you send an updateAccount request, certain account property values must be populated. For more information, see the accountCreate Required Properties table.

Parameters

updateAccount Parameter Descriptions

Parameter Name	Data Type	Description
account	accountUpdate	Data object

Response

None

1.5.2.2.1.3 readAccount

Gets requested properties for an existing account given its unique key.

Parameters

readAccount Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting account record.

Response

readAccount Response Descriptions

Parameter Name	Data Type	Description
account	account	An account record. Only specified property values are returned with the record.

See Also:

account for more information about the properties you can retrieve for the invoice

1.5.2.2.1.4 readAccountsByCriteria

Retrieves a set of accounts based on specified search criteria. If the criteria is empty, then the search will retrieve all accounts. The specified limit takes precedence over the current system setting for the maximum number of search results. For a list of available properties, see the type class, account.

Parameters

readAccountsByCriteria Parameter Descriptions

Parameter Name	Data Type	Description
criteria	searchCriteria	Search criteria or conditions to search for.
limit	int	A numeric value for the maximum number of records to return among search results.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting account records.

Response

readAccountsByCriteria Response Descriptions

Parameter Name	Data Type	Description
accounts	List<account>	A list of account objects. Only specified property values are returned with the record.

See Also:

account for more information about the properties you can retrieve for the account

1.5.2.2.1.5 readChildAccounts

Retrieves child accounts and their requested properties for a given parent account's unique key.

Parameters

readChildAccounts Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the parent account record whose child accounts to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting account records.

Response

readChildAccounts Response Descriptions

Parameter Name	Data Type	Description
account	account	An account record. Only specified property values are returned with the record.

See Also:

account for more information about the properties you can retrieve for the invoice

1.5.2.2.1.6 activateAccount

Activates an account record.

Parameters

activateAccount Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the account record to activate.

Response

None

1.5.2.2.1.7 deactivateAccount

Deactivates an account record.

Parameters

deactivateAccount Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the account record to deactivate.

Response

None

1.5.2.2.1.8 allocateMoney

Deposits funds to an account.

Parameters

allocateMoney Parameter Descriptions

Parameter Name	Data Type	Description
accountUniqueKey	string	Unique key of the record to read.
amount	decimal	Monetary amount to deposit to the account.
description	String	Description of the transaction.

Response

None

1.5.2.2.1.9 transferMoney

Transfers funds between accounts.

Parameters

transferMoney Parameter Descriptions

Parameter Name	Data Type	Description
fromAccountUniqueKey	string	Unique key of the account record to transfer money from.
toAccountUniqueKey	string	Unique key of the account record to transfer money to.
amount	decimal	Monetary amount to transfer between the accounts.
description	String	Description of the transaction.

Response

None

1.5.2.2.1.10 withdrawMoney

Withdraws funds from a given account.

Parameters

withdrawMoney Parameter Descriptions

Parameter Name	Data Type	Description
accountUniqueKey	string	Unique key of the record to withdraw funds from.
amount	decimal	Monetary amount to withdraw.
description	String	Description of the transaction.

Response

None

1.5.2.2.1.11 deleteAccount

Deletes the account specified by the unique key.

Parameters

deleteAccount Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to delete.

Response

There is no return value.

1.5.2.2.1.12 readRecentlyViewedAccounts

Read and return recently viewed accounts.

Parameters

readRecentlyViewedAccounts Parameter Descriptions

Parameter Name	Data Type	Description
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting account records.

Response

readRecentlyViewedAccounts Response Descriptions

Parameter Name	Data Type	Description
accounts	List<account>	A list of account objects. Only specified property values are returned with the record.

1.5.2.2.2 AppointmentRepository Method Details

This section describes the AppointmentRepository methods, including parameters and return values. For the method to insert a new record, a list of required properties is provided.

1.5.2.2.2.1 insertAppointment

Inserts or creates an appointment record.

Parameters

insertAppointment Parameter Descriptions

Parameter Name	Data Type	Description
appointment	appointmentCreate	Data object

Required Properties from appointmentCreate

The following table lists properties from the accountCreate data object that must be populated in order to create an account. Also note that the same properties must be populated when you are updating an account, using the accountUpdate data object.

appointmentCreate Required Properties

Property Name	Data Type	Description
subject	string	The name or reason for the appointment. This field supports up to 250 characters.
startOn	dateTime	The appointment start date and time.
endOn	dateTime	The appointment start date and time.
attendeeCreates	List<attendeeCreate>	Add one or more attendees to an appointment using this list of attendeeCreate data objects.
categories	List<category>	Enable one or more categories using this list of categories. ***Note:*** *The categories you identify must already be defined in TeamConnect through the Appointment Object Definition.*

Response

insertAppointment Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the newly created appointment record.

See Also:

appointmentCreate for more information about the properties you can set for the appointment

1.5.2.2.2.2 updateAppointment

Updates an appointment record.

When you send an updateAppointment request, certain appointment property values must be populated. For more information, see appointmentCreate Required Properties.

Parameters

updateAppointment Parameter Descriptions

Parameter Name	Data Type	Description
appointment	appointmentUpdate	Data object

Response

None

See Also:

appointmentUpdate for more information about the properties you can set for the appointment

1.5.2.2.2.3 readAppointment

Gets requested properties for an existing appointment given its unique key.

Parameters

readAppointment Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting appointment record.

Response

readAppointment Response Descriptions

Parameter Name	Data Type	Description
appointment	appointment	An appointment record. Only specified property values are returned with the record.

See Also:

appointment for more information about the properties you can retrieve for the appointment

1.5.2.2.2.4 readAppointmentsByCriteria

Retrieves a set of appointments based on specified search criteria. If the criteria is empty, then the search will retrieve all appointments. The specified limit takes precedence over the current system setting for the maximum number of search results. For a list of available properties, see the type class, appointment.

Parameters

readAppointmentsByCriteria Parameter Descriptions

Parameter Name	Data Type	Description
criteria	searchCriteria	Search criteria or conditions to search for.
limit	int	A numeric value for the maximum number of records to return among search results.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting appointment records.

Response

readAppointmentsByCriteria Response Descriptions

Parameter Name	Data Type	Description
appointments	List<appointment>	A list of appointment objects. Only specified property values are returned with the record.

See Also:

appointment for more information about the properties you can retrieve

1.5.2.2.2.5 deleteAppointment

Deletes the appointment specified by the unique key.

Parameters

deleteAppointment Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to delete.

Response

There is no return value.

1.5.2.2.2.6 readRecentlyViewedAppointments

Read and return recently viewed appointments.

Parameters

readRecentlyViewedAppointments Parameter Descriptions

Parameter Name	Data Type	Description
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting appointment records.

Response

readRecentlyViewedAppointments Response Descriptions

Parameter Name	Data Type	Description
appointments	List<appointment>	A list of appointment objects. Only specified property values are returned with the record.

1.5.2.2.3 ContactRepository Method Details

This section describes the ContactRepository methods, including parameters and return values. For the method to insert a new record, a list of required properties is provided.

1.5.2.2.3.1 insertContact

Inserts or creates a contact record.

Parameters

insertContact Parameter Descriptions

Parameter Name	Data Type	Description
contact	ContactCreate	Data object

Required Properties from personCreate

The following table lists properties from the personCreate data object that must be populated in order to create a person. Also note that the same properties must be populated when you are updating a person, using the personUpdate data object.

personCreate Required Properties

Property Name	Data Type	Description
firstName	string	The first name of the person.
lastName	string	The last name of the person.

Required Properties from companyCreate

The following table lists properties from the companyCreate data object that must be populated in order to create a company. Also note that the same properties must be populated when you are updating a company, using the companyUpdate data object.

companyCreate Required Properties

Property Name	Data Type	Description
name	string	The name of the organization.

Response

insertContact Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the newly created contact record.

See Also:

For more information about the properties you can set for the contact, see:

- contactCreate
- CompanyCreate
- PersonCreate

1.5.2.2.3.2 updateContact

Updates a contact record.

When you send an updateContact request, certain contact property values must already exist. For more information, see personCreate Required Properties and companyCreate Required Properties.

Note: *To update a value that does not exist for a contact, use a Create data object to insert the value. For example, if an existing contact does not have a phone number, use ContPhoneNumberCreate to insert a new phone number. After that, use ContPhoneNumberUpdate to update the phone number.*

Parameters

updateContact Parameter Descriptions

Parameter Name	Data Type	Description
contact	ContactUpdate	Data object

Response

None

See Also:

For more information about the properties you can set for the contact, see:

- contactUpdate
- CompanyUpdate
- PersonUpdate

1.5.2.2.3.3 readContact

Gets requested properties for an existing contact given its unique key.

Parameters

readContact Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting contact record.

Response

readContact Response Descriptions

Parameter Name	Data Type	Description
contact	Contact	A contact record. Only specified property values are returned with the record.

See Also:

contact for more information about the properties you can retrieve for the contact

1.5.2.2.3.4 readContactsByCriteria

Retrieves a set of contacts based on specified search criteria. If the criteria is empty, then the search will retrieve all contacts. The specified limit takes precedence over the current system setting for the maximum number of search results. For a list of available properties, see the type class, contact.

Parameters

readContactsByCriteria Parameter Descriptions

Parameter Name	Data Type	Description
criteria	searchCriteria	Search criteria or conditions to search for.
limit	int	A numeric value for the maximum number of records to return among search results.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting contact records.

Response

readContactsByCriteria Response Descriptions

Parameter Name	Data Type	Description
contacts	List<Contact>	A list of contact objects. Only specified property values are returned with the record.

See Also:

contact for more information about the properties you can retrieve

1.5.2.2.3.5 deleteContact

Deletes the contact specified by the unique key.

Parameters

deleteContact Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to delete.

Response

There is no return value.

1.5.2.2.3.6 ReadRecentlyViewedContacts

Read and return recently viewed contacts.

Parameters

readRecentlyViewedContacts Parameter Descriptions

Parameter Name	Data Type	Description
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting contact records.

Response

readRecentlyViewedContacts Response Descriptions

Parameter Name	Data Type	Description
contacts	List<contact>	A list of contact objects. Only specified property values are returned with the record.

1.5.2.2.4 DocumentRepository Method Details

This section describes the DocumentRepository methods, including parameters and return values. For the method to insert a new record, a list of required properties is provided.

1.5.2.2.4.1 insertDocument

Inserts or creates a document record (file).

Depending on the type of document you want to create, use one of the following requests:

- For document files, use the insertDocument request with an input parameter (data object of type documentCreate) that defines the properties of the document to create.
- For shortcuts, use createShortcut with input parameters that define which existing document to create a shortcut to (documentUniqueKey) and the target folder location for the shortcut (parentFolderUniqueKey).
- For subfolders, use createSubFolder with input parameters that define the new folder's name (name) and the target folder location for the shortcut (parentUniqueKey).

Separate requests are provided for creating shortcuts and subfolders.

Parameters

insertDocument Parameter Descriptions

Parameter Name	Data Type	Description
documentCreate	documentCreate	Data object

Required Properties from documentCreate

The following table lists properties from the documentCreate data object that must be populated in order to upload a document file. Also note that the same properties must be populated when you are updating a document, using the documentUpdate data object.

documentCreate Required Properties

Property Name	Data Type	Description
name	string	The name of the document. For example, the file name.
contentTypeUniqueKey	string	The file format. The following values can be entered: • Adobe Acrobat • Audio(aiff) • Audio(wav) • Binary or bin • Data Mapping File • Gif • HTM

		• HTML • Image(jpg) • Image(pcx) • Image(tiff) • JAR File • JSP • JavaClass • JavaScriptCode • Microsoft Access • Microsoft Excel • Microsoft Power Point • Microsoft Word • Quicktime Video • Rich Text Format • Standard Email • Text • Video(mpg) • XML • Zip
content	base64Binary	The binary content of a file.
type	documentType	See documentType for more information.
parentFolderUniqueKey	string	The unique key of the document folder that this file will be located in.

Response

insertDocument Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the newly created document record.

See Also:

documentCreate for more information about the properties you can set for a document

1.5.2.2.4.2 createShortcut

Inserts or creates a shortcut to an existing TeamConnect document.

Parameters

createShortcut Parameter Descriptions

Parameter Name	Data Type	Description
documentUniqueKey	string	Unique key of the document to create a shortcut for.
parentFolderUniqueKey	string	Unique key of the folder to create a shortcut with.

Response

createShortcut Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the shortcut.

1.5.2.2.4.3 createSubFolder

Inserts or creates a subfolder in a given parent folder.

Parameters

createSubFolder Parameter Descriptions

Parameter Name	Data Type	Description
name	string	Name of the folder to create.
parentUniqueKey	string	Unique key of the folder to create a subfolder within.

Response

createSubFolder Response Descriptions

Parameter Name	Data Type	Description

uniqueKey	string	The unique key of the subfolder.

1.5.2.2.4.4 createHyperlinkDefaultCategory

Creates a hyperlink in a given parent folder. The hyperlink category is set to the default category for the Document object definition.

Parameters

createHyperlinkDefaultCategory Parameter Descriptions

Parameter Name	Data Type	Description
parentUniqueKey	string	Unique key of the folder to create a subfolder within.
url	string	The website URL that the hyperlink will link to.
name	string	Name of the hyperlink. For example, in the application UI, the hyperlink name displays but the URL does not.
contactUniqueKey	string	Unique key of the contact record for the contact who creates the hyperlink.

Response

createHyperlinkDefaultCategory Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the hyperlink.

1.5.2.2.4.5 createHyperlink

Creates a hyperlink in a given parent folder. The hyperlink category is set to a specified Document category.

Parameters

createHyperlink Parameter Descriptions

Parameter Name	Data Type	Description

parentUniqueKey	string	Unique key of the folder to create a subfolder within.
url	string	The website URL that the hyperlink will link to.
name	string	Name of the hyperlink. For example, in the application UI, the hyperlink name displays but the URL does not.
documentCategoryUniqueKey	string	Unique key of the existing Document category to set as the hyperlink category.
contactUniqueKey	string	Unique key of the contact record for the contact who creates the hyperlink.

Response

createHyperlink Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the hyperlink.

1.5.2.2.4.6 updateDocument

Updates a document record (file, hyperlink, shortcut, or folder).

When you send an updateDocument request, certain document property values must be populated. For more information, see documentCreate Required Properties.

Note: *Although it is not customary in this guide to provide detailed documentation of the data objects, it should be noted that the shortcutToDocumentUniqueKey property of the documentUpdate data object is the unique key of the shortcut to the current document that you're updating.*

Parameters

updateDocument Parameter Descriptions

Parameter Name	Data Type	Description
document	documentUpdate	Data object

Response

None

1.5.2.2.4.7 readDocument

Gets requested properties for an existing document given its unique key.

Parameters

readDocument Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting document record.

Response

readDocument Response Descriptions

Parameter Name	Data Type	Description
document	document	A document record. Only specified property values are returned with the record.

1.5.2.2.4.8 readDocumentsByCriteria

Retrieves a set of documents based on specified search criteria. If the criteria is empty, then the search will retrieve all documents. The specified limit takes precedence over the current system setting for the maximum number of search results. For a list of available properties, see the type class, document.

Parameters

readDocumentsByCriteria Parameter Descriptions

Parameter Name	Data Type	Description
criteria	searchCriteria	Search criteria or conditions to search for.
limit	int	A numeric value for the maximum number of records to return among search results.

properties	List<String>	A list of String values that identify the property names whose values to return with the resulting invoice records.

Response

readDocumentsByCriteria Response Descriptions

Parameter Name	Data Type	Description
documents	List<document>	A list of document objects. Only specified property values are returned with the record.

1.5.2.2.4.9 readDocumentByPath

Gets a document data object and the specified property values for a given path.

Parameters

readDocumentByPath Parameter Descriptions

Parameter Name	Data Type	Description
path	string	Path to the target document, including the document file name and extension. For example: Top Level\Attachments\Disputes\DISP-000018 \IntakeRecord-3372.doc
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting document.

Response

readDocumentByPath Response Descriptions

Parameter Name	Data Type	Description
document	document	A document record. Only specified property values are returned with the record.

1.5.2.2.4.10 readChildDocuments

This request can perform two functions based on the document unique key parameter.

- If you pass the unique key of a document file (parent), the response includes older document versions of that file.
- If you pass the unique key of a folder, the response includes documents within that folder.

Parameters

readChildDocuments Parameter Descriptions

Parameter Name	Data Type	Description
parentUniqueKey	string	Either the unique key of the document record whose children (previous document versions) to return or the unique key of a folder.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting document.

Response

readChildDocuments Response Descriptions

Parameter Name	Data Type	Description
documents	List<document>	• If the parentUniqueKey was a document file unique key, the list of documents includes previous versions of the given document. • If the parentUniqueKey was a document folder unique key, the list of documents includes the files, folders, shortcuts, and hyperlinks in the given folder. A document record. Only specified property values are returned with the record.

1.5.2.2.4.11 readChildDocumentForName

Gets requested properties for an existing document given the unique key of its parent folder and the document's file name.

Parameters

readChildDocumentForName Parameter Descriptions

Parameter Name	Data Type	Description

uniqueKey	string	The unique key of the target document's parent folder.
name	string	The document file name.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting document.

Response

readChildDocumentForName Response Descriptions

Parameter Name	Data Type	Description
documents	document	A document record. Only specified property values are returned with the record.

1.5.2.2.4.12 readDocumentFolderForDocumentOwner

Gets requested properties for an existing document folder given the unique key of its owner (user).

Parameters

readDocumentFolderForDocumentOwner Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the document folder owner (user).
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting document.

Response

readDocumentFolderForDocumentOwner Response Descriptions

Parameter Name	Data Type	Description
documents	document	A document record. Only specified property values are returned with the record.

1.5.2.2.4.13 checkIn

Checks in a document that had been previously checked out.

When a new version of a document is checked in, its unique key does not change; however, the previous version of the document is saved under a new unique key.

Parameters

checkIn Parameter Descriptions

Parameter Name	Data Type	Description
documentUniqueKey	string	The unique key of the document file. This should be the same unique key passed when you checked out the document.
content	base64Binary	The document file content.
versionText	string	The version number to assign to this document.

Response

None

See Also:

checkOut, revert for related methods

1.5.2.2.4.14 checkOut

Checks out a document. This function is typically used for version control of a file in TeamConnect and to lock a file while it is being edited.

Parameters

checkOut Parameter Descriptions

Parameter Name	Data Type	Description
documentUniqueKey	string	The unique key of the document file to check out.

Response

None

See Also:

checkIn, revert for information about related methods

1.5.2.2.4.15 undoCheckOut

Attempts to undo check out of a given document.

Parameters

undoCheckOut Parameter Descriptions

Parameter Name	Data Type	Description
documentUniqueKey	string	The unique key of a document record that has been checked out.

Response

None

See Also:

checkOut for information about related methods

1.5.2.2.4.16 revert

To revert a document to an older document version, you need to know the unique key of that older document version. You can accomplish this by getting a list of older document versions by calling the readChildDocuments method, then find the older version to revert back to from the list.

Parameters

revert Parameter Descriptions

Parameter Name	Data Type	Description
documentUniqueKey	string	Document file unique key that identifies the version of the document (for example the current version) to replace.
oldKey	string	Document file unique key that identifies the version of the document to revert to (sets this file to the current version).

Response

None

See Also:

checkOut, checkIn for information about related records

1.5.2.2.4.17 moveDocument

Moves a document to a new parent folder.

Parameters

moveDocument Parameter Descriptions

Parameter Name	Data Type	Description
documentUniqueKey	string	The unique key of the document record to move.
newParentFolderUniqueKey	string	The unique key of the folder where the target document should be moved.

Response

None

1.5.2.2.4.18 copyDocument

Copies a document to a new parent folder.

Parameters

copyDocument Parameter Descriptions

Parameter Name	Data Type	Description
documentUniqueKey	string	The unique key of the document record to copied.
newParentFolderUniqueKey	string	The unique key of the folder where the document copy should be created.

Response

None

1.5.2.2.4.19 deleteDocument

Deletes the persistent document specified by the unique key.

Parameters

deleteDocument Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to delete.

Response

There is no return value.

1.5.2.2.4.20 ReadRecentlyViewedDocuments

Read and return recently viewed documents.

Parameters

readRecentlyViewedDocuments Parameter Descriptions

Parameter Name	Data Type	Description
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting document records.

Response

readRecentlyViewedContacts Response Descriptions

Parameter Name	Data Type	Description
documents	List<document>	A list of document objects. Only specified property values are returned with the record.

1.5.2.2.4.21 setDocumentAsRecentlyViewed

Set document to recently viewed status.

Parameters

setDocumentAsRecentlyViewed Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	String	Unique key of the document record set as recently viewed.

Response

There is no return value.

1.5.2.2.4.22 readFolderHierarchy

Retrieves a hierarchal list of document folders for one or more projects.

Parameters

readFolderHierarchy Parameter Descriptions

Parameter Name	Data Type	Description
projUniqueKey	List<string>	A list of unique keys for each project.
properties	List<string>	A list of strings that identify the property names of the values to return for each folder.

Response

readFolderHierarchy Response Descriptions

Parameter Name	Data Type	Description
readFolderHierarchy	List<DocumentTreeNode>	A list of tree nodes. Each node includes the property values for the folders in the project and specifies the hierarchy of the folders. This parameter only returns specified property values.

1.5.2.2.5 ExpenseRepository Method Details

This section describes the ExpenseRepository methods, including parameters and return values. For the method to insert a new record, a list of required properties is provided.

1.5.2.2.5.1 insertExpense

Inserts or creates an expense record.

Parameters

insertExpense Parameter Descriptions

Parameter Name	Data Type	Description
expense	expenseCreate	Data object

Required Properties from expenseCreate

The following table lists properties from the expenseCreate data object that must be populated in order to upload an expense. Also note that the same properties must be populated when you are updating an expense, using the expenseUpdate data object.

expenseCreate Required Properties

Property Name	Data Type	Description
shortDescription	string	The expense description.
expenseDate	date	The expense date. For example, when the expense was incurred or the date the record was made.
expensedByUniqueKey	string	The unique key of the user responsible for the expense.

Response

insertExpense Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the newly created record.

See Also:

expenseCreate for more information about the properties you can set for the expense

1.5.2.2.5.2 updateExpense

Update an expense record.

When you send an updateExpense request, certain expense property values must be populated. For more information about the required properties when creating or updating expenses, see the expenseCreate Required Properties table.

Parameters

updateExpense Parameter Descriptions

Parameter Name	Data Type	Description
expense	expenseUpdate	Data object

Response

There is no return value.

See Also:

expenseUpdate for more information about the properties you can change for expenses

1.5.2.2.5.3 readExpense

Gets requested properties for an existing expense given its unique key.

Parameters

readExpense Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting expense record.

Response

readExpense Response Descriptions

Parameter Name	Data Type	Description
expense	expense	An expense record. Only specified property values are returned with the record.

See Also:

expense for more information about the properties you can retrieve for the invoice

1.5.2.2.5.4 readExpensesByCriteria

Retrieves a list of expenses based on specified search criteria. If the criteria is empty, then the search will retrieve all expenses. The specified limit takes precedence over the current system setting for the maximum number of search results. For a list of available properties, see the type class, expense.

Parameters

readExpensesByCriteria Parameter Descriptions

Parameter Name	Data Type	Description
criteria	searchCriteria	Search criteria or conditions to search for.
limit	int	A numeric value for the maximum number of records to return among search results.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting expense records.

Response

readExpensesByCriteria Response Descriptions

Parameter Name	Data Type	Description
expenses	List<expense>	A list of expense objects. Only specified property values are returned with the record.

See Also:

expense for more information about the properties you can retrieve for the expense

1.5.2.2.5.5 postExpense

Posts a given expense.

Parameters

postExpense Parameter Descriptions

Parameter Name	Data Type	Description

uniqueKey	string	Unique key of the record to post.

Response

There is no return value.

1.5.2.2.5.6 voidExpense

Voids a given expense.

Parameters

voidExpense Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to void.

Response

There is no return value.

1.5.2.2.5.7 deleteExpense

Deletes the expense specified by the unique key.

Parameters

deleteExpense Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to delete.

Response

There is no return value.

1.5.2.2.5.8 readRecentlyViewedExpenses

Read and return recently viewed expenses.

Parameters

readRecentlyViewedDocuments Parameter Descriptions

Parameter Name	Data Type	Description
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting expense records.

Response

readRecentlyViewedContacts Response Descriptions

Parameter Name	Data Type	Description
expenses	List<expense>	A list of expense objects. Only specified property values are returned with the record.

1.5.2.2.6 GroupAccountRepository Method Details

This section describes the GroupAccountRepository methods, including parameters and return values. For the method to insert a new record, a list of required properties is provided.

1.5.2.2.6.1 insertGroupAccount

Inserts or creates a group account record.

Parameters

insertGroupAccount Parameter Descriptions

Parameter Name	Data Type	Description
groupAccount	groupAccountCreate	Data object

Required Properties from groupAccountCreate

The following table lists properties from the groupAccountCreate data object that must be populated in order to upload a groupAccount. Also note that the same properties must be populated when you are updating a groupAccount, using the groupAccountUpdate data object.

groupAccountCreate Required Properties

Property Name	Data Type	Description
uniqueName	string	A unique name for the group. Values can include letters and numbers. Special characters are not allowed.

displayName	string	The group name. Maximum length: 50 characters.
userAccountUniqueKeys	List<String>	A list of unique key values for the user accounts to add to the group.

Response

insertGroupAccount Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the newly created group account record.

See Also:

groupAccountCreate for more information about the properties you can set for the group

1.5.2.2.6.2 updateGroupAccount

Updates a group account record.

When you send an updateGroupAccount request, certain contact property values must be populated. For more information, see groupAccountCreate Required Properties.

Parameters

updateGroupAccount Parameter Descriptions

Parameter Name	Data Type	Description
groupAccount	groupAccountUpdate	Data object

Response

None

See Also:

groupAccountUpdate for more information about the properties you can set for the user

1.5.2.2.6.3 readGroupAccount

Gets requested properties for an existing group account given its unique key.

Parameters

readGroupAccount Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting group account record.

Response

readGroupAccount Response Descriptions

Parameter Name	Data Type	Description
groupAccount	groupAccount	A group account record. Only specified property values are returned with the record.

See Also:

groupAccount for more information about the properties you can retrieve for the group

1.5.2.2.6.4 readGroupAccountsByCriteria

Retrieves a set of group accounts based on specified search criteria. If the criteria is empty, then the search will retrieve all group accounts. The specified limit takes precedence over the current system setting for the maximum number of search results. For a list of available properties, see the type class, groupAccount.

Parameters

readGroupAccountsByCriteria Parameter Descriptions

Parameter Name	Data Type	Description
criteria	searchCriteria	Search criteria or conditions to search for.
limit	int	A numeric value for the maximum number of records to return among search results.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting group account records.

Response

readGroupAccountsByCriteria Response Descriptions

Parameter Name	Data Type	Description
groupAccounts	List<groupAccount>	A list of group account objects. Only specified property values are returned with the record.

See Also:

groupAccount for more information about the properties you can retrieve

1.5.2.2.6.5 deleteGroupAccount

Deletes the group account specified by the unique key.

Parameters

deleteGroupAccount Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to delete.

Response

There is no return value.

1.5.2.2.6.6 readRecentlyViewedGroupAccounts

Read and return recently viewed accounts.

Parameters

readRecentlyViewedDocuments Parameter Descriptions

Parameter Name	Data Type	Description
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting account records.

Response

readRecentlyViewedContacts Response Descriptions

Parameter Name	Data Type	Description
groupAccounts	List<accounts>	A list of account objects. Only specified property values are returned with the record.

1.5.2.2.7 HistoryRepository Method Details

This section describes the HistoryRepository methods, including parameters and return values. For the method to insert a new record, a list of required properties is provided.

1.5.2.2.7.1 insertHistory

Inserts or creates a history record. Although a history is a unique and searchable record, you will typically associate a history with another record (for example, an account), using the history to track changes made to the account.

Parameters

insertHistory Parameter Descriptions

Parameter Name	Data Type	Description
history	historyCreate	Data object

Required Properties from historyCreate

The following table lists properties from the historyCreate data object that must be populated in order to create a history record. Also note that the same properties must be populated when you are updating a history record, using the historyUpdate data object.

historyCreate Required Properties

Property Name	Data Type	Description
archivedOn	dateTime	The date the history record is created. For example, the current date.
shortDescription	string	Description of the history or note about the related record's changes. Maximum: 250 characters.
parentObject	historyOwner	The unique key of the record to associate the history record with.

Response

insertHistory Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the newly created history record.

See Also:

historyCreate for more information about the properties you can set for the history

1.5.2.2.7.2 updateHistory

Updates a history record.

When you send an updateHistory request, certain history property values must be populated. For more information, see historyCreate Required Properties.

Parameters

updateHistory Parameter Descriptions

Parameter Name	Data Type	Description
history	historyUpdate	Data object

Response

None

See Also:

historyUpdate for more information about the properties you can set for the account

1.5.2.2.7.3 readHistory

Gets requested properties for an existing history given its unique key.

Parameters

readHistory Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting history record.

Response

readHistory Response Descriptions

Parameter Name	Data Type	Description
history	history	A history record. Only specified property values are returned with the record.

See Also:

history for more information about the properties you can retrieve for the record

1.5.2.2.7.4 readHistoriesByCriteria

Retrieves a set of history records based on specified search criteria. If the criteria is empty, then the search will retrieve all history records. The specified limit takes precedence over the current system setting for the maximum number of search results. For a list of available properties, see the type class, history.

Parameters

readHistoriesByCriteria Parameter Descriptions

Parameter Name	Data Type	Description
criteria	searchCriteria	Search criteria or conditions to search for.
limit	int	A numeric value for the maximum number of records to return among search results.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting history records.

Response

readHistoriesByCriteria Response Descriptions

Parameter Name	Data Type	Description
histories	List<history>	A list of history objects. Only specified property values are returned with the record.

See Also:

history for more information about the properties you can retrieve for the record

1.5.2.2.7.5 deleteHistory

Deletes the history record specified by the unique key.

Parameters

deleteHistory Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to delete.

Response

There is no return value.

1.5.2.2.7.6 readRecentlyViewedHistories

Read and return recently view history records.

Parameters

readRecentlyViewedHistories Parameter Descriptions

Parameter Name	Data Type	Description
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting history records.

Response

readRecentlyViewedHistories Response Descriptions

Parameter Name	Data Type	Description
histories	List<history>	A list of history objects. Only specified property values are returned with the record.

1.5.2.2.8 InvoiceRepository Method Details

This section describes the InvoiceRepository methods, including parameters and return values. For the method to insert a new record, a list of required properties is provided.

1.5.2.2.8.1 insertInvoice

Inserts or creates an invoice record.

To create an invoice, use the insertInvoice request with an input parameter (data object of type InvoiceCreate) that defines the properties of the invoice to create.

The response to creating an invoice is the unique key value of the new record. The unique key uniquely identifies a record and is used as a parameter in operations like updating and reading records.

Typically when approval rules (workflow) have been created for invoices, there may be scenarios when an approver rejects an invoice and you may resubmit another version of an existing invoice. When you create or update an invoice that has the same number, date, and vendor as an existing rejected invoice, the new or updated invoice will be saved with an auto-incremented version. For example, if the original rejected invoice's version was 1.0 then the resubmitted invoice's version will automatically be set to 1.1. For more information about rejected invoices, see the User Guide section on Invoices.

Parameters

insertInvoice Parameter Descriptions

Parameter Name	Data Type	Description
invoiceCreate	invoiceCreate	Data object

Required Properties from invoiceCreate

The following table lists properties from the invoiceCreate data object that must be populated in order to create an invoice. Also note that the same properties must be populated when you are updating an invoice, using the invoiceUpdate data object.

invoiceCreate Required Properties

Property Name	Data Type	Description
numberString	string	The invoice number.
invoiceDate	date	The invoice date. For example, the date the invoice was submitted.
vendorUniqueKey	string	The account start date.

Required Properties from lineItemCreate

The following table lists properties from the lineItemCreate data object that must be populated in order to create a line item. Although you can create or update an invoice with no line items, this is very impractical. Note that there are different required properties depending on the type of line item you are creating or editing.

Also note that the same properties must be populated when you are updating an invoice, using the invoiceUpdate data object with the lineItemUpdate data object.

lineItemCreate Required Properties

Property Name	Data Type	Description
itemNumber	int	Numeric ID that will set the display order of the line item in the application UI.
serviceDate	date	date for the expense or services provided.
projectUniqueKey	string	The unique key of the project associated with this line item.
categories	List<category>	• If the type property is expense, the category should be a line item expense category. • If the type property is fee, the category should be a line item task category.
type	lineItemType	The line item type (EXPENSE or FEE).
timekeeperUniqueKey	string	This property is only required if the type is Fee. Unique key of the timekeeper's contact record.
activityUniqueKey	string	This property is only required if the type is Fee. Tree position of the item from the Activity Item system lookup table to associate with the Fee/Task.
originalRate	decimal	• If the specified type is expense, the unit price for an expense. • If the specified type is fee, the original rate for services rendered.
originalQuantity	decimal	• If the specified type is expense, the number of units for an expense. • If the specified type is fee, the number of hours of a service.

Response

insertInvoice Response Descriptions

Parameter Name	Data Type	Description

uniqueKey	string	The unique key of the newly created invoice record.

See Also:

invoiceCreate for more information about the properties you can set for the invoice

lineItemCreate for more information about the properties you can set for line items

1.5.2.2.8.2 updateInvoice

Update an invoice record.

When you send an updateInvoice request, certain invoice property values must be populated. For more information about the required properties when creating or updating line items, see the invoiceCreate Required Properties table. In addition, when updating an invoice it is most practical for the invoice to include at least one line item. For more information about the required properties when creating or updating line items, see the lineItemCreate Required Properties table.

Parameters

updateInvoice Parameter Descriptions

Parameter Name	Data Type	Description
invoice	invoiceUpdate	Data object

Response

There is no return value.

See Also:

invoiceUpdate for more information about the properties you can change for invoices

lineItemUpdate for more information about the properties you can set for line items

1.5.2.2.8.3 readInvoice

Gets requested properties for an existing invoice given its unique key.

Parameters

readInvoice Parameter Descriptions

Parameter Name	Data Type	Description

uniqueKey	string	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting invoice record.

Response

readInvoice Response Descriptions

Parameter Name	Data Type	Description
invoice	invoice	An invoice record. Only specified property values are returned with the record.

See Also:

invoice for more information about the properties you can retrieve for the invoice

lineItem for more information about the properties you can retrieve for line items

1.5.2.2.8.4 readInvoicesByCriteria

Retrieves a list of invoices based on specified search criteria. If the criteria is empty, then the search will retrieve all invoices. The specified limit takes precedence over the current system setting for the maximum number of search results. For a list of available properties, see the type class, invoice.

Parameters

readInvoicesByCriteria Parameter Descriptions

Parameter Name	Data Type	Description
criteria	searchCriteria	Search criteria or conditions to search for.
limit	int	A numeric value for the maximum number of records to return among search results.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting invoice records.

Response

readInvoicesByCriteria Response Descriptions

Parameter Name	Data Type	Description
invoices	List<invoice>	A list of invoice objects. Only specified property values are returned with the record.

See Also:

invoice for more information about the properties you can retrieve for the invoice

lineItem for more information about the properties you can retrieve for line items

1.5.2.2.8.5 postInvoice

Posts a given invoice.

Parameters

postInvoice Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to post.

Response

There is no return value.

1.5.2.2.8.6 voidInvoice

Voids a given invoice.

Parameters

voidInvoice Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to void.

Response

There is no return value.

1.5.2.2.8.7 deleteInvoice

Deletes the invoice specified by the unique key.

Parameters

deleteInvoice Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to delete.

Response

There is no return value.

1.5.2.2.8.8 readRecentlyViewedInvoices

Parameters

readRecentlyViewedInvoices Parameter Descriptions

Parameter Name	Data Type	Description
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting invoice records.

Response

readRecentlyViewedInvoices Response Descriptions

Parameter Name	Data Type	Description
invoices	List<invoice>	A list of invoice objects. Only specified property values are returned with the record.

1.5.2.2.8.9 adjustInvoiceHeader

Parameters

adjustInvoiceHeader Parameter Descriptions

Parameter Name	Data Type	Description

invoiceUniqueKey	String	Unique key of the record to adjust.
method	adjustmentMethod	Method for adjustment (REDUCE_BY_AMOUNT, REDUCE_BY_PERCENT, or NEW_AMOUNT).
target	invoiceAdjustmentTarget	Invoice target to be adjusted (TOTAL_FEES, TOTAL_EXPENSES, or TOTAL_INVOICE).
value	decimal	The amount for the adjustment.
reasonKey	string	Tree position of the line item adjustment reasons table.
commentsToVendor	string	Comments to the vendor.
internalComments	string	Internal comments for the adjustment.

Response

There is no return value.

1.5.2.2.8.10 readActiveApprovals

Parameters

readActiveApprovals Parameter Descriptions

Parameter Name	Data Type	Description
invoiceUniqueKey	String	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting approval records.

Response

readActiveApprovals Response Descriptions

Parameter Name	Data Type	Description
approvals	List<approval>	A list of approval objects. Only specified property values are returned with the record.

1.5.2.2.8.11 readCompletedApprovals

Parameters

readCompletedApprovals Parameter Descriptions

Parameter Name	Data Type	Description
invoiceUniqueKey	String	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting approval records.

Response

readCompletedApprovals Response Descriptions

Parameter Name	Data Type	Description
approvals	List<approval>	A list of approval objects. Only specified property values are returned with the record.

1.5.2.2.8.12 readInvoiceApprovalsPendingOnPost

Read all pending invoice approvals that are pending on post.

Parameters

readInvoiceApprovalsPendingOnPost Parameter Descriptions

Parameter Name	Data Type	Description
limit	int	A numeric value for the maximum number of records to return among search results.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting approval records.

Response

readInvoiceApprovalsPendingOnPost Response Descriptions

Parameter Name	Data Type	Description

approvals	List<approval>	A list of approval objects. Only specified property values are returned with the record.

1.5.2.2.9 LookupTableSource Method Details

This section describes the LookupTableSource methods, including parameters and return values.

1.5.2.2.9.1 readSystemLookupTable

Retrieves a list of properties for a system table code. You use the properties to read values from lookup tables.

Parameters

readSystemLookup Parameter Descriptions

Parameter Name	Data Type	Description
SystemTableCode	string	Unique four-letter code for the system table types. Refer to the following list of codes for each type: • RESO—Appointment resource • ADDR—Contact address • MAIL—Contact email address • INET—Contact internet address • PHON—Contact phone number • FAXX—Contact fax number • CONR—Contact relation • SKIL—Contact skill • TERR—Contact territory • ACTI—Line item task activity • LTAR—Line item adjustment reason • PRJR—Project relation • COUN—Country • STAT—State • CURR—Currency • NUST—Invoice non-US tax • MLTT—Matter level tax

Response

readSystemLookup Response Descriptions

Parameter Name	Data Type	Description
WSLookupTable	WSLookupTable	An item that includes a list of properties for a lookup table item. The list includes the following properties: • Item name • Order item displays • Stored value of the item • Active status of the item • Subtypes of item

1.5.2.2.10 ProjectRepository Method Details

This section describes the ProjectRepository methods, including parameters and return values. For the method to insert a new record, a list of required properties is provided.

1.5.2.2.10.1 insertProject

Inserts or creates a project record.

Parameters

insertProject Parameter Descriptions

Parameter Name	Data Type	Description
projectCreate	projectCreate	Data object

Required Properties from projectCreate

The following table lists properties from the projectCreate data object that must be populated in order to create a project. Also note that the same properties must be populated when you are updating a project, using the projectUpdate data object.

projectCreate Required Properties

Property Name	Data Type	Description
entityTypeUniqueKey	string	Unique 4-digit alphanumeric code that identifies the custom object. Corresponds to the TeamConnect Unique Code value.

name	string	The project record name.
idNumber	string	The project record ID number.
categories	List<Category>	The categories to add or enable for the project record.

Response

insertProject Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the newly created project record.

See Also:

projectCreate for more information about the properties you can set for the project

projectAssigneeCreate for more information about adding assignees to the project and the corresponding assignee properties you can set

projRelationCreate for more information about adding relations to the project and the corresponding relation properties you can set

1.5.2.2.10.2 updateProject

Updates a project or matter record.

When you send an updateProject request, certain project property values must be populated. For more information, see the projectCreate Required Properties table.

Parameters

updateProject Parameter Descriptions

Parameter Name	Data Type	Description
projectUpdate	projectUpdate	Data object

Response

None

See Also:

projectUpdate for more information about the properties you can set for the invoice

projectAssigneeUpdate for more information about the properties you can update for assignees

projRelationUpdate for more information about the properties you can update for relations

1.5.2.2.10.3 readProject

Gets requested properties for an existing project given its unique key.

Parameters

readProject Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting project record.

Response

readProject Response Descriptions

Parameter Name	Data Type	Description
project	project	A project record. Only specified property values are returned with the record.

See Also:

project for more information about the properties you can retrieve

projectAssignee for more information about the properties you can retrieve for assignees

projRelation for more information about the properties you can retrieve for relations

phaseInterval for more information about the properties you can retrieve for phases

1.5.2.2.10.4 readProjectsByCriteria

Retrieves a set of projects based on specified search criteria. If the criteria is empty, then the search will retrieve all projects. The specified limit takes precedence over the current system setting for the maximum number of search results. For a list of available properties, see the type class, project.

Parameters

readProjectsByCriteria Parameter Descriptions

Parameter Name	Data Type	Description
criteria	searchCriteria	Search criteria or conditions to search for.
limit	int	A numeric value for the maximum number of records to return among search results.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting invoice records.

Response

readProjectsByCriteria Response Descriptions

Parameter Name	Data Type	Description
projects	List<project>	A list of project objects. Only specified property values are returned with the record.

See Also:

project for more information about the properties you can retrieve

projectAssignee for more information about the properties you can retrieve for assignees

projRelation for more information about the properties you can retrieve for relations

phaseInterval for more information about the properties you can retrieve for phases

1.5.2.2.10.5 readChildProjectsForEntityType

Retrieves requested properties of child project records for a given parent project record. You must also specify the project type of child projects to retrieve.

Parameters

readChildProjectsForEntityType Parameter Descriptions

Parameter Name	Data Type	Description
parentProjectUniqueKey	string	Unique key of the parent project record whose child project records to read.
projectEntityTypeUniqueCode	string	Unique code or tree position for the default category of the child project records to return.

		The unique code will be used like a search criteria and child projects that have the specified unique code set as default will be returned.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting projects.

Response

readChildProjectsForEntityType Response Descriptions

Parameter Name	Data Type	Description
project	project	A project record. Only specified property values are returned with the record.

See Also:

project for more information about the properties you can retrieve

projectAssignee for more information about the properties you can retrieve for assignees

projRelation for more information about the properties you can retrieve for relations

phaseInterval for more information about the properties you can retrieve for phases

1.5.2.2.10.6 changePhase

Changes a project records phase. You must get the valid phase's unique key from the TeamConnect application interface.

Parameters

changePhase Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the project whose phase to change.
phaseUniqueKey	string	Unique code of the phase to set for the project. ***Note:*** *You need to know beforehand the defined phase intervals (for supported phase transitions) and phase unique codes.*

Response

None

1.5.2.2.10.7 deleteProject

Deletes the project specified by the unique key.

Parameters

deleteProject Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to delete.

Response

None

1.5.2.2.10.8 readRecentlyViewedProjects

Parameters

readRecentlyViewedProjects Parameter Descriptions

Parameter Name	Data Type	Description
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting project records.

Response

readRecentlyViewedProjects Response Descriptions

Parameter Name	Data Type	Description
projects	List<project>	A list of project objects. Only specified property values are returned with the record.

1.5.2.2.10.9 readProjectEntityTypes

Get a list of project entity types in TeamConnect.

Parameters

readProjectEntityTypes Parameter Descriptions

Parameter Name	Data Type	Description
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting project entity type records.

Response

readProjectEntityTypes Response Descriptions

Parameter Name	Data Type	Description
projectEntityTypes	List<projectEntityType>	A list of project entity types. Only specified property entity type values are returned with the record.

1.5.2.2.10.10 readProjectIntegrationSearches

Get a list of search views that are of type "Integration". Each returned search view should have a unique key, a name, and its associated entity type.

Response

readProjectIntegrationSearches Response Descriptions

Parameter Name	Data Type	Description
projectSearches	List<projectSearch>	A list of project searches that are of type "Integration".

1.5.2.2.10.11 readProjectsUsingSearch

Parameters

readProjectsUsingSearch Parameter Descriptions

Parameter Name	Data Type	Description
projectEntityTypeUniqueKey	string	Unique 4-digit alphanumeric code that identifies the custom object. Corresponds to the TeamConnect Unique Code value.
searchUniqueKey	string	Unique key of the project search of type "Integration".

limit	int	A numeric value for the maximum number of records to return among search results.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting project records.

Response

readProjectEntityTypes Response Descriptions

Parameter Name	Data Type	Description
projects	List<project>	A list of project objects. Only specified property values are returned with the record.

1.5.2.2.11 InvolvedRepository Method Details

This section describes the InvolvedRepository methods, including parameters and return values. For the method to insert a new record, a list of required properties is provided.

1.5.2.2.11.1 insertInvolved

Inserts or creates an involved party record. Note that when an involved record is created with the active status set to true, the activeDate will automatically be set to the current date.

Parameters

insertInvolved Parameter Descriptions

Parameter Name	Data Type	Description
involved	involvedCreate	Data object

Required Properties from involvedCreate

The following table lists properties from the involvedCreate data object that must be populated in order to create an involved party. Also note that the same properties must be populated when you are updating an involved party, using the involvedUpdate data object.

involvedCreate Required Properties

Property Name	Data Type	Description

entityTypeUniqueKey	string	Unique 4-digit alphanumeric code that identifies the involved child object (child of associated project object definition). Corresponds to the TeamConnect Unique Code value.
projectUniqueKey	string	Project record unique key that the involved party should be associated with.
contactUniqueKey	string	Contact record unique key for the involved party.
active	boolean	True indicates the involved party is active.
categories	List<Category>	The categories to add or enable for the involved party record.

Response

insertInvolved Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the newly created involved party record.

See Also:

involvedUpdate for more information about the properties you can update for the involved record

1.5.2.2.11.2 updateInvolved

Updates an involved record.

Parameters

updateInvolved Parameter Descriptions

Parameter Name	Data Type	Description
involved	involvedUpdate	Data object

Response

None

See Also:

involvedUpdate for more information about the properties you can update for the involved record

1.5.2.2.11.3 readInvolved

Gets requested properties for an existing involved record given its unique key.

Parameters

readInvolved Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting project record.

Response

readInvolved Response Descriptions

Parameter Name	Data Type	Description
involved	involved	An involved record. Only specified property values are returned with the record.

See Also:

involved for more information about the properties you can retrieve

1.5.2.2.11.4 readInvolvedsForProject

Gets requested properties for involved party records that are associated with a project record, given the unique key of the project record.

Parameters

readInvolvedsForProject Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the project record whose associated involved party records to read.

properties	List<String>	A list of String values that identify the property names whose values to return with the resulting involved party records.

Response

readInvolvedsForProject Response Descriptions

Parameter Name	Data Type	Description
involved	List<involved>	A list of involved records. Only specified property values are returned with the records.

See Also:

involved for more information about the properties you can retrieve

1.5.2.2.11.5 readInvolvedsByCriteria

Gets requested properties for involved party records given search criteria.

Parameters

readInvolvedsByCriteria Parameter Descriptions

Parameter Name	Data Type	Description
involvedEntityTypeUniqueCode	string	Unique code of the involved party object definition whose records to read. This should correspond to the involved party that is a child object of associated target project.
criteria	searchCriteria	Search criteria or conditions to search for.
limit	int	The maximum number of involved party records to return in the search results.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting involved party records.

Response

readInvolvedsByCriteria Response Descriptions

Parameter Name	Data Type	Description
involved	List<involved>	A list of involved party records. Only specified property values are returned with the records.

See Also:

involved for more information about the properties you can retrieve

1.5.2.2.11.6 deleteInvolved

Deletes an involved part record given its unique key.

Parameters

deleteInvolved Parameter Description

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to read.

Response

None

1.5.2.2.12 TaskRepository Method Details

This section describes the TaskRepository methods, including parameters and return values. For the method to insert a new record, a list of required properties is provided.

1.5.2.2.12.1 insertTask

Inserts or creates a task record.

Parameters

insertTask Parameter Descriptions

Parameter Name	Data Type	Description
task	taskCreate	Data object

Required Properties from taskCreate

The following table lists properties from the taskCreate data object that must be populated in order to create a task record. Also note that the same properties must be populated when you are updating a task record, using the taskUpdate data object.

taskCreate Required Properties

Property Name	Data Type	Description
shortDescription	string	A brief description of the task.
currentAssignee	taskAssigneeCreate	The user assigned to the task.

Response

insertTask Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the newly created record.

See Also:

taskCreate for more information about the properties you can set for the expense

1.5.2.2.12.2 updateTask

Update a task record.

When you send an updateTask request, certain task property values must be populated. For more information about the required properties when creating or updating tasks, see Table 2.192 "taskCreate Required Properties".

Parameters

updateTask Parameter Descriptions

Parameter Name	Data Type	Description
task	taskUpdate	Data object

Response

There is no return value.

See Also:

taskUpdate for more information about the properties you can change for tasks

1.5.2.2.12.3 readTask

Gets requested properties for an existing task given its unique key.

Parameters

readTask Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting task record.

Response

readTask Response Descriptions

Parameter Name	Data Type	Description
task	task	A task record. Only specified property values are returned with the record.

See Also:

task for more information about the properties you can retrieve for the task

1.5.2.2.12.4 readTasksByCriteria

Retrieves a list of tasks based on specified search criteria. If the criteria is empty, then the search will retrieve all tasks. The specified limit takes precedence over the current system setting for the maximum number of search results. For a list of available properties, see the type class, task.

Parameters

readTasksByCriteria Parameter Descriptions

Parameter Name	Data Type	Description
criteria	searchCriteria	Search criteria or conditions to search for.
limit	int	A numeric value for the maximum number of records to return among search results.

properties	List<String>	A list of String values that identify the property names whose values to return with the resulting expense records.

Response

readTasksByCriteria Response Descriptions

Parameter Name	Data Type	Description
tasks	List<task>	A list of task objects. Only specified property values are returned with the record.

See Also:

task for more information about the properties you can retrieve for the task

1.5.2.2.12.5 postTask

Posts a given task.

Parameters

postTask Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to post.

Response

There is no return value.

1.5.2.2.12.6 voidTask

Voids a given task.

Parameters

voidTask Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to void.

Response

There is no return value.

1.5.2.2.12.7 reassign

Reassigns a given task to a different user.

Parameters

reassign Parameter Descriptions

Parameter Name	Data Type	Description
taskUniqueKey	string	Unique key of the task to reassign.
userUniqueKey	string	Unique key of the user to set as the new task assignee.

Response

There is no return value.

1.5.2.2.12.8 deleteTask

Deletes the task specified by the unique key.

Parameters

deleteTask Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to delete.

Response

There is no return value.

1.5.2.2.12.9 readRecentlyViewedTasks

Read recently viewed tasks.

Parameters

readRecentlyViewedTasks Parameter Descriptions

Parameter Name	Data Type	Description
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting task records.

Response

readRecentlyViewedTasks Response Descriptions

Parameter Name	Data Type	Description
tasks	List<task>	A list of task objects. Only specified property values are returned with the record.

1.5.2.2.13 UserAccountRepository Method Details

This section describes the UserAccountRepository methods, including parameters and return values. For the method to insert a new record, a list of required properties is provided.

1.5.2.2.13.1 insertUserAccount

Inserts or creates a user account record.

Parameters

insertUserAccount Parameter Descriptions

Parameter Name	Data Type	Description
userAccount	userAccountCreate	Data object

Required Properties from userAccountCreate

The following table lists properties from the userAccountCreate data object that must be populated in order to create a user account record. Also note that the same properties must be populated when you are updating a user account record, using the userAccountUpdate data object.

userAccountCreate Required Properties

Property Name	Data Type	Description
username	string	The user name (this will be used to log into the application UI). Maximum length: 50 characters.

password	string	The user's password. Maximum length: 250 characters
contactUniqueKey	string	The corresponding contact record's unique key. Each user must have an associated contact record.
userType	userType	Although this property is not required to create a user account, it is highly recommended to populate it. • Super—Can access all public and private records. This setting overrides access restrictions based on record-level security settings. Recommended for system administrators and solution developers. • Normal—Can access all public records. Can access private records only if created by the user or if record-level security is granted to the user. Recommended for internal staff. • Limited-Privilege—Can access only records created by the user, records where the user is an assignee or attendee, and records with record-level security granted to the user. Recommended for Outside Counsel. ***Note:*** *Access to a record automatically grants access to its embedded records but not to its related records. For example, if a limited-privilege user has the right to access a dispute record, they cannot automatically access the dispute's related involved parties, tasks, histories, or expenses, but they can access its allegation (embedded object) records.*
active	boolean	Although this property is not required to create a user account, it is highly recommended to populate it. Status of account activation (true indicates the account is active).
groupAccountAddUniqueKeys	List<String>	Although this property is not required to create a user account, it is highly recommended to populate it.

		List of unique keys for group records that the current user should be added to.

Response

insertUserAccount Response Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	The unique key of the newly created user account record.

See Also:

userAccountCreate for more information about the properties you can set for the user

userType for more information about the properties you can set for the user

1.5.2.2.13.2 updateUserAccount

Updates a user account record.

When you send an updateUserAccount request, certain contact property values must be populated. For more information, see the userAccountCreate Required Properties table.

Parameters

updateUserAccount Parameter Descriptions

Parameter Name	Data Type	Description
userAccount	userAccountUpdate	Data object

Response

None

See Also:

userAccountUpdate for more information about the properties you can set for the user

1.5.2.2.13.3 readUserAccount

Gets requested properties for an existing user account given its unique key.

Parameters

readUserAccount Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to read.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting user account record.

Response

readUserAccount Response Descriptions

Parameter Name	Data Type	Description
userAccount	userAccount	A user account record. Only specified property values are returned with the record.

See Also:

userAccount for more information about the properties you can retrieve for the user

1.5.2.2.13.4 readUserAccountsByCriteria

Retrieves a set of user accounts based on specified search criteria. If the criteria is empty, then the search will retrieve all user accounts. The specified limit takes precedence over the current system setting for the maximum number of search results. For a list of available properties, see the type class, userAccount.

Parameters

readUserAccountsByCriteria Parameter Descriptions

Parameter Name	Data Type	Description
criteria	searchCriteria	Search criteria or conditions to search for.
limit	int	A numeric value for the maximum number of records to return among search results.
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting user account records.

Response

readUserAccountsByCriteria Response Descriptions

Parameter Name	Data Type	Description
userAccounts	List<userAccount>	A list of user account objects. Only specified property values are returned with the record.

See Also:

userAccount for more information about the properties you can retrieve

1.5.2.2.13.5 deleteUserAccount

Deletes the user account specified by the unique key.

Parameters

deleteUserAccount Parameter Descriptions

Parameter Name	Data Type	Description
uniqueKey	string	Unique key of the record to delete.

Response

There is no return value.

1.5.2.2.13.6 readRecentlyViewedUserAccounts

Read recently viewed user accounts.

Parameters

readRecentlyViewedUserAccounts Parameter Descriptions

Parameter Name	Data Type	Description
properties	List<String>	A list of String values that identify the property names whose values to return with the resulting user account records.

Response

readRecentlyViewedUserAccounts Response Descriptions

Parameter Name	Data Type	Description
userAccounts	List<userAccount>	A list of user account objects. Only specified property values are returned with the record.

1.5.2.3 REST API HTTP Methods

You can use JSON formatted HTTP requests to create, read, update, or delete records from the TeamConnect server using the REST API methods. You must have the rights to view or make changes to these records.

Before you use the REST API HTTP methods, enter the following information:

- `http://<host>/<TeamConnectVersion>/webservice/rest` for the root of the API.
- `application/json` for the content-type and accept headers.
- HTTP basic authentication with TeamConnect credentials.

The HTTP methods are GET, POST, PUT, and DELETE. When you make a request using one of the HTTP methods, refer to the following tables.

HTTP Methods for Records

HTTP Method	Format	Description
GET	(nothing after GET)	Returns a list of record types.
	`/<TypeOfRecord>` Example: `/tasks`	Returns all the records of that type in the server. To limit the number of results, enter a number in the **Number of records per result page** field on the **General** page of the **Preferences** in TeamConnect. To see additional pages of results, enter ?page=2, or another number depending on the page, after /<TypeOfRecord>. To search text or checkbox fields, enter the following: `?<TreePostition>.<FieldName>=<value> after /<TypeOfRecord>, for example, ?DISP_SUBP.DocketNoSU=1234`
	`/<TypeOfRecord>/<ID>` Example: `/tasks/2`	Returns information about the record.
POST	`/<TypeOfRecord>`	Creates a new record. To determine the structure of the JSON object, refer to Determining the structure of a JSON object.

	/<TypeOfRecord>/<ID>	Returns an error.
PUT	/<TypeOfRecord>	Returns an error.
	/<TypeOfRecord>/<ID>	Updates a record. To determine the structure of the JSON object, refer to Determining the structure of a JSON object.
DELETE	/<TypeOfRecord>	Returns an error.
	/<TypeOfRecord>/<ID>	Deletes a record.

HTTP Methods for Projects

HTTP Method	Format	Description
GET	(nothing after GET)	Returns a list of projects and project codes.
	/projects/<ProjectCode> Example: /projects/DISP	Returns all the records of that project type in the server.
	/projects/<ProjectCode>/<ID> Example: /projects/DISP/3	Returns information about the project record.
POST	/projects/<ProjectCode>	Creates a new project record. To determine the structure of the JSON object, refer to Determining the structure of a JSON object.
	/projects/<ProjectCode>/<ID>	Returns an error.
PUT	/projects/<ProjectCode>	Returns an error.
	/projects/<ProjectCode>/<ID>	Updates a project record. To determine the structure of the JSON object, refer to Determining the structure of a JSON object.

DELETE	/projects/<ProjectCode>	Returns an error.
	/projects/<ProjectCode>/<ID>	Deletes a project record.

Determining the structure of a JSON object

When you create or update a new record or project using the POST or PUT method, use the same format as a JSON object. Use the GET method with a record type and ID to return a formatted example.

The following structure is an example of a JSON object for /tasks:

```
{
    "shortDescription":"Created using the API",
    "currentAssignee": {
    "assignedOn":"Aug 15, 2012 8:15:00 AM", {
    "href": "/user/15"
    }
    }
}
```

The following structure is an example of a JSON object for /appointments:

```
{
    "subject":"Created using the API",
    "startOn":"Aug 20, 2012 4:31:00 PM",
    "endOn":"Aug 20, 2012 5:31:00 PM",
    "attendees":[ {
    "href": "/user/3"
    }]
}
```

The following structure is an example of a JSON object for /DISP:

```
{
    "entityType":"DISP",
    "name":"Created using the API",
    "categories": {
    "DISP":{
    "categoryItem":"DISP",
    "customFields":{}
    }
        "DISP_BANK":{
        "categoryItem":"DISP_BANK",
        "customFields":{}
        }
    }
}
```

Troubleshooting Error Messages

You receive an error message if you enter the following:

- An invalid username or password.
- Invalid parameter values for a record.
- An ID that does not exist.
- No ID when you enter a REST method that requires an ID (PUT and DELETE).
- An ID when the REST method does not require an ID (POST).

You might also receive an error message if you do not enter what the method requires. For example, if you are trying to update a task with the PUT method, but you do not enter new values, you receive an error message.

1.5.2.4 Abstract Classes

The following table summarizes abstract classes used in TeamConnect Web Services.

Abstract Classes Summary

Abstract Class Name	Description and Subclasses
applicationEntityUpdate	Provides a property, uniqueKey, with corresponding setter method. Use the uniqueKey setter method to identify which record to update. Subclasses include: • accountUpdate • appointmentUpdate • contactUpdate • documentUpdate • embeddedEntityUpdate • expenseUpdate • groupAccountUpdate • historyUpdate • invoiceUpdate • involvedUpdate • lineItemUpdate • projectUpdate • taskUpdate • userAccountUpdate

	Note: *The subclasses of applicationEntityUpdate specify additional properties with corresponding setter methods for updating record-specific properties.*
lookupItem	Provides a group of properties and corresponding getter methods that you use to read values from TeamConnect system lookup tables, custom lookup tables, and the multi-currency lookup table. Subclasses include: • appointmentResourceType • contAddressType • countryItem • contEmailAddressType • contFaxNumberType • contInetAddressType • contPhoneNumberType • contRelationType • contSkillType • contTerritoryItem • currencyItem (property of contact and invoice) • invoiceNonUSTaxType • lineItemAdjustmentReason • projRelationType • taskActivityItem ***Note:*** *The setter methods associated with lookupItem should not be used.* ***Note:*** *The subclasses listed above have related Create and Update classes (for example, contAddressCreate and contAddressUpdate). Use the setTypeUniqueKey method, setting the value to the existing system lookup table item's Tree Position, to populate the corresponding type (for example contAddressType).*
contact	Provides properties and corresponding getter methods for reading or searching contacts. When defining a contact repository search or

	read operation's return properties, refer to the property names defined in this object. Subclasses include: • Person • Company
contactCreate	Provides properties and corresponding setter methods to populate a contact's properties. Subclasses include: • PersonCreate • CompanyCreate
contactUpdate	Provides properties and corresponding setter methods to update or clear a contact's properties. Subclasses include: • PersonUpdate • CompanyUpdate

1.5.2.5 Category and Custom Field Data Objects

The following table summarizes data objects used to add/remove categories to records or populate/update/clear custom field values in records using TeamConnect Web Services.

Categories and Custom Field Data Objects

Data Object	Description
booleanCustomField	Provides a method for identifying the existing boolean custom field (property) to update (populate). Also provides a method to set the value of a boolean custom field.
dateTimeCustomField	Provides a method for identifying the existing date/time custom field (property) to update (populate). Also provides a method to set the value of a date/time custom field.
decimalCustomField	Provides a method for identifying the existing decimal custom field (property) to update (populate). Also provides a method to set the value of a decimal custom field.
involvedCustomField	Provides a method for identifying the existing involved party custom field (property) to update (populate). Also provides a method to set the value of a involved party custom field.

noteCustomField	Provides a method for identifying the existing note custom field (property) to update (populate). Also provides a method to set the value of a note custom field.
projectCustomField	Provides a method for identifying the existing project custom field (property) to update (populate). Also provides a method to set the value of a project custom field.
textCustomField	Provides a method for identifying the existing text custom field (property) to update (populate). Also provides a method to set the value of a text custom field.

1.5.2.6 Data Objects Used in Search Criteria

There are a set of data objects that you use to define search criteria when you are searching records. This section provides a summary for data objects used in searches.

Data Objects Used in Search Criteria

Data Object	Description
fieldSearchClause	Subclass of searchCriteria. Provides methods to add field criteria to a search definition. Also provides a method to set the logic for combining multiple field criterion in a search.
legacySearchFieldPathExpression	Subclass of fieldPathExpression. Provides a method to set a searchKeyPath or define the TeamConnect field (Web Service property) whose value you will search by.
booleanComparator	Use this enumeration to set the method for comparing a boolean field value against existing values in the TeamConnect database.
dateComparator	Use this enumeration to set the method for comparing a date field value against existing values in the TeamConnect database. Use this with both calendarDateFieldCriterion and dateFieldCriterion.
decimalComparator	Use this enumeration to set the method for comparing a decimal field value against existing values in the TeamConnect database.
documentFolderComparator	Use this enumeration to set the method for comparing a document folder field value against existing values in the TeamConnect database.

enumerationComparator	Use this enumeration to set the method for comparing an enumeration field value against existing values in the TeamConnect database.
integerComparator	Use this enumeration to set the method for comparing an integer field value against existing values in the TeamConnect database.
objectComparator	Use this enumeration to set the method for comparing an object field value against existing values in the TeamConnect database.
relativeDateComparator	Use this enumeration to set the method for comparing a date field value against existing values in the TeamConnect database when using the relativeDateFieldCriterion object.
stringComparator	Use this enumeration to set the method for comparing a string field value against existing values in the TeamConnect database.
userComparator	Use this enumeration to set the method for comparing a user field value against existing values in the TeamConnect database.
booleanFieldCriterion	Use this to define search criteria that filters search results according to a boolean type field value.
calendarDateFieldCriterion	Use this to define search criteria that filters search results according to a date (time zone dependent) field value.
dateFieldCriterion	Use this to define search criteria that filters search results according to a date (time zone independent) field value.
decimalFieldCriterion	Use this to define search criteria that filters search results according to a decimal field value.
documentFolderFieldCriterion	Use this to define search criteria that filters search results according to a document folder field value.
enumerationFieldCriterion	Use this to define search criteria that filters search results according to a field value of type, enumeration. For example, for searching documents by documentType (enumeration) property value.

integerFieldCriterion	Use this to define search criteria that filters search results according to a integer field value.
objectFieldCriterion	Use this to define search criteria that filters search results according to a field value of type, object.
relativeDateFieldCriterion	Use this to define search criteria that filters search results according to a relative date field value.
stringFieldCriterion	Use this to define search criteria that filters search results according to a string field value.
userFieldCriterion	Use this to define search criteria that filters search results according to a field value of type, user.

1.5.3 Frequently Asked Questions

This section provides tips for using Web Services.

1.5.3.1 Batch Operations

When using Web Services, you need to make a unique repository request per record creation, record update, record read, records search, and record deletion operation.

Adding or updating certain record property values can be performed in a batch manner. For example:

- Contact addresses, email addresses, internet addresses
- Contact phone numbers, fax numbers
- Contact rates, relations, skills, territories
- Invoice adjustments
- Invoice line items, and line item adjustments
- Project assignees, relations
- Task assignees

1.5.3.2 Operations Not Supported In Web Services

The following operations must be performed from the TeamConnect UI:

- Setting the default category for a record

  When a record is created through Web Services, its default category is automatically set as the root category. To change a record's default category, use the TeamConnect UI.

- Creating, updating, reading, deleting object definitions
- Creating, updating, reading, deleting categories

- Creating or deleting custom fields
- Creating, updating, reading, deleting phases
- Creating, updating, reading, deleting phase transitions
- Creating, updating, reading, deleting rules

1.5.3.3 Cosmetic Data Objects

In the Web Service types.xsd file, you may see certain data objects that are used internally by Web Services but are not useful when you write a client program. As a result, you can ignore the following data objects (which may be auto-generated as data objects or classes when you run a SOAP toolkit against the Web Service source files):

- user
- userCreate
- userUpdate
- phaseIntervalCreate
- phaseIntervalUpdate
- involvedEntityType
- applicationEntityType
- projectEntityType
- applicationEntityType
- visibility
- directive
- fieldType
- systemTableCode

Note: *If you want to work with the User record type, the following data objects are available: userAccount, userAccountCreate, and userAccountUpdate.*

1.5.4 Client Application Components (Java/Apache CXF)

This appendix contains sample code for the following components of a client application:

- Client Proxy
  - Web Service client interface classpath reference
  - Web Service URL
- Security Headers for system authentication
- Client Application that combines instantiation of the Client Proxy and insertion of the interceptor for Security Headers addition

Note: *All samples were written in JAVA for use with the Apache CXF.*

1.5.4.1 Client Proxy Sample Code

The following code sample illustrates concepts described in Client Proxy of the Getting Started section.

It is important to note that the JaxWsProxyFactoryBean or equivalent class should be provided by your SOAP toolkit to create a client proxy. In addition, note that you can use the entityName (object type name, such as Contact) to construct both the classpath to the corresponding Web Service client interface, and to construct the corresponding Web Service URL.

Note: *In this sample, the TeamConnect version is presumed to be 3.2.*

```
package com.mitratech.teamconnect;
import org.apache.commons.lang.StringUtils;
import org.apache.cxf.jaxws.JaxWsProxyFactoryBean;
/**
 * The purpose of this class is to create a Web Service client side proxy
 *
 * @author
 * @since 3.0
 */
public class WSClientFactory
{
    private final static String SERVER = "localhost";
    private final static String PORT = "8080";
    private final static String APP_NAME = "teamconnect-3.4SP1.0.0021";
    /**
     *
     * @param entityName
     * @return a service proxy for the specified entity
     */
    public Object createWSProxy(String entityName) {
        JaxWsProxyFactoryBean clientFactory = new JaxWsProxyFactoryBean();
        try {
            clientFactory.setServiceClass(getClassForEntity(entityName));
        } catch (ClassNotFoundException cnfe) {
            throw new IllegalArgumentException("No Web Services available for entity
            named: " + entityName);
        }
        clientFactory.setAddress(getWSAddress(entityName));
        return clientFactory.create();
    }
    private Class getClassForEntity(String entityName) throws
    ClassNotFoundException {
```

```
        return Class.forName("com.mitratech.teamconnect.webservice." +
        entityName.toLowerCase() + "repository." + entityName + "Repository");
    }
    private String getWSAddress(String entityName) {
        return "http://" + SERVER + ":" + PORT + "/" + APP_NAME + "/webservice/" +
        StringUtils.uncapitalize(entityName) + "Repository";
    }
}
```

1.5.4.2 Security Headers Sample Code

The following code sample illustrates concepts described in SOAP Message Security Header of the Getting Started section.

It is important to know that the TeamConnect user name and password need to be stored as String variables. Also note that in the resulting SOAP request message, the password will be transferred in cleartext format (unencrypted).

```
package com.mitratech.teamconnect;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import org.apache.cxf.endpoint.Client;
import org.apache.cxf.endpoint.Endpoint;
import org.apache.cxf.frontend.ClientProxy;
import org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor;
import org.apache.ws.security.WSConstants;
import org.apache.ws.security.WSPasswordCallback;
import org.apache.ws.security.handler.WSHandlerConstants;
public class WSSecurityHeadersHelper
{
    private final static String USER = "system";
    private final static String PASSWORD = "s";
    public static void applySecurityHeaders(final Object sei) {
        Client cxfClient = ClientProxy.getClient(sei);
        Endpoint cxfEndpoint = cxfClient.getEndpoint();
        Map<String, Object> outProps = new HashMap<String, Object>();
        outProps.put(WSHandlerConstants.ACTION, WSHandlerConstants.USERNAME_TOKEN);
        outProps.put(WSHandlerConstants.USER, USER);
        outProps.put(WSHandlerConstants.PASSWORD_TYPE, WSConstants.PW_TEXT);
        outProps.put(WSHandlerConstants.PW_CALLBACK_CLASS,
        ClientPasswordCallback.class.getName());
        WSS4JOutInterceptor wssOut = new WSS4JOutInterceptor(outProps);
        cxfEndpoint.getOutInterceptors().add(wssOut);
    }
```

```
    public static class ClientPasswordCallback implements CallbackHandler
    {
        public void handle(Callback[] callbacks) throws IOException,
        UnsupportedCallbackException {
            WSPasswordCallback pc = (WSPasswordCallback) callbacks[0];
            pc.setPassword(PASSWORD);
        }
    }
}
```

1.5.4.3 Client Application Sample Code

The following code sample illustrates concepts described in Putting Components Together in a Client Application of the Getting Started section. The code is an excerpt from a client application that creates a client proxy for the contactRepository.

It is important to note that when you instantiate the client proxy using the createWSProxy method, you would type cast the resulting object according to the entityName you had passed in. For example, if you passed in the entityName, Contact, then you'd type cast the returned object as ContactRepository. For the entityName, Document, then you'd type cast the returned object as DocumentRepository.

It is also important to note that you add the interceptor for the message security headers only one time to the client proxy. Afterward, when messages are generated for the client application requests, the corresponding security headers will be automatically inserted in each message. See the code like: WSSecurityHeadersHelper.applySecurityHeaders(contactRepository)

```
package com.mitratech.teamconnect;
import java.util.Date;
import java.util.List;
import com.mitratech.teamconnect.webservice.contactrepository.ContactRepository;
import com.mitratech.teamconnect.webservice.type.Category;
import com.mitratech.teamconnect.webservice.type.ContAddressCreate;
import com.mitratech.teamconnect.webservice.type.ContAddressUpdate;
import com.mitratech.teamconnect.webservice.type.Contact;
import com.mitratech.teamconnect.webservice.type.FieldSearchClause;
import com.mitratech.teamconnect.webservice.type.LegacySearchFieldPathExpression;
import com.mitratech.teamconnect.webservice.type.LogicOperator;
import com.mitratech.teamconnect.webservice.type.Person;
import com.mitratech.teamconnect.webservice.type.PersonCreate;
import com.mitratech.teamconnect.webservice.type.PersonUpdate;
import com.mitratech.teamconnect.webservice.type.StringComparator;
import com.mitratech.teamconnect.webservice.type.StringFieldCriterion;
import com.mitratech.teamconnect.webservice.type.TextCustomField;
public class WSContactClient extends AbstractWSClient
{
    private ContactRepository contactRepository;
    public WSContactClient() {
        contactRepository = (ContactRepository)createWSProxy("Contact");
```

```
        WSSecurityHeadersHelper.applySecurityHeaders(contactRepository);
        fieldNames.add("firstName");
        fieldNames.add("addresses.uniqueKey");
        fieldNames.add("addresses.type.name");
        fieldNames.add("addresses.city");
        fieldNames.add("addresses.state");
        fieldNames.add("addresses.countryCode.name");
        fieldNames.add("categories");
    }
    public static void main(String[] args) throws Exception {
        WSContactClient contactClient = new WSContactClient();
        String uniqueKey = contactClient.insertContact();
        Person person = contactClient.readContactByKey(uniqueKey);
        contactClient.updateContact(uniqueKey);
        person = contactClient.readContactByKey(uniqueKey);
        List<Contact> contacts =
        contactClient.readContactsByFirstName(person.getFirstName());
        contacts = contactClient.readContactsByPrimaryLanguage("English");
        contacts = contactClient.readContactsByPrimaryLanguage("French");
        contactClient.deletePerson(uniqueKey);
        person = contactClient.readContactByKey(uniqueKey);
        System.out.println("done");
    }
    private String insertContact() throws Exception {
        String uniqueKey = contactRepository.insertContact(createPerson());
        return uniqueKey;
    }
    private Person readContactByKey(String uniqueKey) throws Exception
    {
        Person person = (Person)contactRepository.readContact(uniqueKey,
        fieldNames);
        return person;
    }
    private PersonCreate createPerson() {
        String now = "" + new Date().getTime();
        PersonCreate person = new PersonCreate();
        person.setFirstName("first name " + now);
        person.getAddressCreates().add(createContAddress());
        ContAddressCreate anotherAddress = createContAddress();
        anotherAddress.setCity("San Jose");
        person.getAddressCreates().add(anotherAddress);
        Category cat = new Category();
        cat.setUniqueKey("CONT");
        TextCustomField primaryLanguage = new TextCustomField();
        primaryLanguage.setFieldName("primaryLanguage");
```

```
        primaryLanguage.setValue("English");
        cat.getTextCustomFields().add(primaryLanguage);
        person.getCategories().add(cat);
        return person;
    }
    private ContAddressCreate createContAddress() {
        ContAddressCreate address = new ContAddressCreate();
        address.setTypeUniqueKey("ADDR_HOME");
        address.setCity("Los Angeles");
        address.setState("CA");
        address.setCountryCodeUniqueKey("COUN_0002");
        return address;
    }
}
```

1.5.5 Code Samples (Java/Apache CXF)

This chapter provides additional code samples (snippets) for using Web Services when writing a JAVA client program and using the Apache CXF SOAP toolkit. Samples for the following Web Services are provided:

- ContactRepository
- AccountRepository
- AppointmentRepository
- DocumentRepository
- ExpenseRepository
- GroupAccountRepository
- HistoryRepository
- InvoiceRepository
- ProjectRepository
- InvolvedRepository
- TaskRepository
- UserAccountRepository

Each TeamConnect Web Service's or repository's sample code is provided by repository name in alphabetic order for ease of navigation. However the object types can be grouped conceptually like the following:

- System Objects—AccountRepository, AppointmentRepository, ContactRepository, ExpenseRepository, InvoiceRepository, TaskRepository
    - DocumentRepository (this type of record can be stand-alone but is commonly associated with another record)

- HistoryRepository (this type of record is always associated with another record, for capturing information about changes to the record information)

- Custom Objects—ProjectRepository
  - InvolvedRepository (although a system object, this type of record is commonly associated with a project record)
- Account Administration Objects—GroupAccountRepository, UserAccountRepository

Basic overview descriptions for common functions like inserting, updating, reading, searching, and deleting records are in the chapter, Getting Started under Common Functions.

Note: *For JAVA code samples provided in this guide, JAX-WS and JAXB methods were used for JAVA XML Binding.*

1.5.5.1 Contacts

The following .JAVA code samples show how you can use TeamConnect® Enterprise Web Services for functions specific to contact records.

In the code samples for creating person and company contacts, the required fields are indicated in comments.

1.5.5.1.1 Creating Contacts

When creating a contact record, you need to further specify the type of contact you are creating. This is done through the contact argument. To specify a person type contact, pass in a contact created through the PersonCreate object. To specify a company type contact, pass in a contact created through the CompanyCreate object. When you instantiate the PersonCreate or CompanyCreate objects, they are used to store contact record properties. Both the PersonCreate and CompanyCreate objects extend ContactCreate, an abstract class.

Code Snippet for creating a contact of type person and populating a property from a system lookup table

```
protected ContactRepository contactRepository;
PersonCreate person = new PersonCreate();
//firstName is a required field
person.setFirstName("John");
//lastName is a required field
person.setLastName("Doe");
ContAddressCreate address = new ContAddressCreate();
address.setTypeUniqueKey("HOME");
//where the typeUniqueKey value, "HOME", is an existing item's Tree Position value
under the system lookup table, Address Type
address.setStreet("2468 Washington Ave.");
address.setCity("Los Angeles");
address.setState("CA");
address.setPostalCode("90026");
```

```
person.getAddressCreates().add(address);
String uniqueKey = contactRespository.insertContact(person);
//when you create a contact, the record's unique key is returned
//alternatively you could create a company contact like
contactRespository.insertContact(company)
//where company would be an instance of the CreateCompany object
```

Code Snippet for creating a contact of type company

```
protected ContactRepository contactRepository;
CompanyCreate company = new CompanyCreate();
//name is a required field
company.setName("ACME Inc.");
//taxNumber is used to identify vendor contacts if you are using CSM/Collaborati
company.setTaxNumber("0112498531");
String uniqueKey = contactRespository.insertContact(company);
//when you create a contact, the record's unique key is returned
```

Code Snippet for adding categories and custom fields to a person (contact)

```
protected ContactRepository contactRepository;
PersonCreate person = createPerson();
String personUniqueKey = contactRepository.insertContact(person);
private PersonCreate createPerson() {
    String firstName = "first" + new Date().getTime();
    PersonCreate person = new PersonCreate();
    person.setFirstName(firstName);
    person.setBirthdate(Calendar.getInstance().getTime());
    ContAddressCreate address = new ContAddressCreate();
    address.setTypeUniqueKey("ADDR_HOME");
    address.setCity("Portland");
    address.setState("OR");
    address.setCountryCodeUniqueKey("COUN_0002");
    person.getAddressCreates().add(address);
    Category cat = new Category();
    //the next two lines identify a category and then add the category to the
    contact record
    cat.setUniqueKey("CONT_EXTE");
    person.getCategories().add(cat);
    Category cat2 = new Category();
    person.getCategories().add(cat2);
    cat2.setUniqueKey("CONT");
    //the next four lines create a placeholder for an existing Boolean custom
    field, identify the known custom field name (synched), set the custom field
    value, and add the custom field placeholder to its parent category
    BooleanCustomField synched = new BooleanCustomField();
    synched.setFieldName("synched");
```

```
synched.setValue(true);
cat2.getBooleanCustomFields().add(synched);
TextCustomField primaryLanguage = new TextCustomField();
primaryLanguage.setFieldName("primaryLanguage");
primaryLanguage.setValue("Englishe US");
cat2.getTextCustomFields().add(primaryLanguage);
return person;
}
```

1.5.5.1.2 Updating Contacts

When updating contacts, use the PersonUpdate or CompanyUpdate objects as parameters to the ContactRepository updateContact method.

Some data associated with a contact record is stored through separate objects. For example, a contact's rates are added through the ContRateCreate object. In addition, contact rates are updated through the ContRateUpdate object. The same concepts apply to a contact's mailing addresses (ContAddressCreate), email addresses (ContEmailAddressCreate), internet addresses (ContInetAddressCreate), phone numbers (ContPhoneNumberCreate), fax numbers (ContFaxNumberCreate), relations (ContRelationCreate), skills (ContSkillCreate), and territories (ContTerritoryCreate).

To delete certain data associated with a contact record (for example, rates, mailing addresses, email addresses, internet addresses, phone numbers, fax numbers, relations, skills, territories), use the following:

```
PersonUpdate.getRateDeleteUniqueKeys().add(newItem);
```

where newItem is the unique key of the rate to delete. Note that you need to repeat the line above for each rate to delete.

Also see the code snippets under Creating Contacts for comments about required contact fields.

Code Snippet for updating existing properties

```
protected ContactRepository contactRepository;
//for this example, it is assumed that you already know the target contact record's unique key (where uniqueKey is a String variable)
PersonUpdate personUpdate = new PersonUpdate();
personUpdate.setUniqueKey(uniqueKey);
personUpdate.setLastName("Doe II");
contactRepository.updateContact(personUpdate);
```

Code Snippet for clearing existing property values

```
protected ContactRepository contactRepository;
//for this example, it is assumed that you already know the target contact record's unique key (where uniqueKey is a String variable)
PersonUpdate personUpdate = new PersonUpdate();
personUpdate.setUniqueKey(uniqueKey);
personUpdate.getClearedProps().addAll(getPropertiesToClear());
private List<String> getPropertiesToClear() {
```

```
    List<String> properties = new List<String>();
    properties.add("firstName");
    properties.add("lastName");
    properties.add("address.type");
    properties.add("address.state");
    return properties;
}
```

Code Snippet for adding contact rates

```
protected ContactRepository contactRepository;
//for this example, it is assumed that you already know the target contact record's unique key (where uniqueKey is a String variable)
PersonUpdate personUpdate = new PersonUpdate();
personUpdate.setUniqueKey(uniqueKey);
ContRateCreate rate = new ContRateCreate();
Date today = new Date();
rate.setFirstEffectiveDate(today);
rate.setLastEffectiveDate(new Date(today.getTime() + 100000));
rate.setRate(new BigDecimal(20));
rate.setTaskCategoryCode("TASK_DEFA");
personUpdate.getRateCreates().add(rate);
contactRepository.updateContact(personUpdate);
```

Code Snippet for updating a contact's custom field values

```
protected ContactRepository contactRepository;
//for this example, it is assumed that you already know the target contact record's unique key (where uniqueKey is a String variable)
PersonUpdate personUpdate = new PersonUpdate();
personUpdate.setUniqueKey(uniqueKey);
Category cat = new Category();
personUpdate.getCategories().add(cat);
//The category should already exist. You should get its unique key (tree position) from TeamConnect.
cat.setUniqueKey("CONT_EXTE");
TextCustomField primaryLanguage = new TextCustomField();
cat.getTextCustomFields().add(primaryLanguage);
primaryLanguage.setFieldName("primaryLanguage");
primaryLanguage.setValue("Chinese PRC");
contactRepository.updateContact(personUpdate);
```

1.5.5.1.3 Reading Contacts

When reading a contact record, identify the target record by unique key. Use the ContactRepository readContact method.

Code Snippet

```
//for this example, it is assumed that you already know the target contact record's unique key (where uniqueKey is a String variable)
protected ContactRepository contactRepository;
public void readPersonContact() throws Exception {
    Person readPerson = (Person) contactRepository.readContact(uniqueKey, getPropertiesToRead());
}
private List<String> getPropertiesToRead() {
    List<String> properties = new List<String>();
    properties.add("firstName");
    properties.add("lastName");
    properties.add("address.type");
    properties.add("address.state");
    return properties;
}
```

1.5.5.1.4 Searching for Contacts

When searching for contacts, use the ContactRepository readContactsByCriteria method. Use the Person, Company, and Contact data objects to reference the field names you can use in the search criteria and include in the properties list for values to return with the search results (contact records). From the search results, the numeric limit value you set determines how many contact records will be returned. Similar to when you are reading a contact record, you must specify the names of property values to return with search results.

For more general information about how to define search criteria, see Searching Records.

Code Snippet for readContactsByCriteria

```
//for this example, it is assumed that at least one contact record that meets the search criterion already exists
protected ContactRepository contactRepository;
public void searchPersonContacts() throws Exception {
    StringFieldCriterion fieldCriterion = new StringFieldCriterion();
    fieldCriterion.setComparator(StringComparator.EQUALS_ENFORCE_CASE);
    LegacySearchFieldPathExpression fieldPathExpression = new LegacySearchFieldPathExpression();
    fieldPathExpression.setSearchKeyPath("firstName");
    fieldCriterion.setFieldPath(fieldPathExpression);
    fieldCriterion.getValue().add("John");
    FieldSearchClause searchCriteria = new FieldSearchClause();
    searchCriteria.setOperator(LogicOperator.AND);
    searchCriteria.getCriteria().add(fieldCriterion);
```

```
    List<Contact> contacts =
    contactRepository.readContactsByCriteria(searchCriteria, 100,
    getPropertiesToRead());
}
private List<String> getPropertiesToRead() {
    List<String> properties = new List<String>();
    properties.add("firstName");
    properties.add("lastName");
    properties.add("addresses.type");
    properties.add("addresses.state");
    return properties;
}
```

Code Snippet for readContactsByCriteria (specify return properties of contact property of type object)

```
//for this example, it is assumed that at least one contact record that meets the
search criterion already exists
//the last three properties defined in the properties list illustrate how you
specify contact search return property values where the contact property is an
object
protected ContactRepository contactRepository;
public void searchPersonContacts() throws Exception {
    StringFieldCriterion fieldCriterion = new StringFieldCriterion();
    fieldCriterion.setComparator(StringComparator.EQUALS_ENFORCE_CASE);
    LegacySearchFieldPathExpression fieldPathExpression = new
    LegacySearchFieldPathExpression();
    fieldPathExpression.setSearchKeyPath("firstName");
    fieldCriterion.setFieldPath(fieldPathExpression);
    fieldCriterion.getValue().add("John");
    FieldSearchClause searchCriteria = new FieldSearchClause();
    searchCriteria.setOperator(LogicOperator.AND);
    searchCriteria.getCriteria().add(fieldCriterion);
    List<Contact> contacts =
    contactRepository.readContactsByCriteria(searchCriteria, 100,
    getPropertiesToRead());
}
private List<String> getPropertiesToRead() {
    List<String> properties = new List<String>();
    properties.add("firstName");
    properties.add("lastName");
    properties.add("primaryPhoneNumber.number");
    //where number is a property of the contact property, primaryPhoneNumber (type
    object)
    properties.add("primaryEmailAddress.address");
    //where address is a property of the contact property, primaryEmailAddress
    (type object)
    properties.add("defaultRates.rate");
```

```
    //where rate is a property of the contact property, defaultRates (type object)
    return properties;
}
```

Code Snippet for readContactsByCriteria (search by custom field value)

```
//for this example, it is assumed that at least one contact record that meets the search criterion already exists
//also assume that a category, Employee (with unique code, EMPL) has been defined for the Contact object definition
//also assume that a custom string field, DLNumb (driver's license number) has been define for the Contact category, Employee
protected ContactRepository contactRepository;
public void searchPersonContacts() throws Exception {
    StringFieldCriterion fieldCriterion = new StringFieldCriterion();
    fieldCriterion.setComparator(StringComparator.EQUALS_ENFORCE_CASE);
    LegacySearchFieldPathExpression fieldPathExpression = new
    LegacySearchFieldPathExpression();
    fieldPathExpression.setSearchKeyPath("CONT_EMPL__DLNumb");
    //note there are two underscores between "EMPL" and "DLNumb" in the
    searchKeyPath above
    fieldCriterion.setFieldPath(fieldPathExpression);
    fieldCriterion.getValue().add("B301792Z");
    FieldSearchClause searchCriteria = new FieldSearchClause();
    searchCriteria.setOperator(LogicOperator.AND);
    searchCriteria.getCriteria().add(fieldCriterion);
    List<Contact> contacts =
    contactRepository.readContactsByCriteria(searchCriteria, 100,
    getPropertiesToRead());
}
private List<String> getPropertiesToRead() {
    List<String> properties = new List<String>();
    properties.add("firstName");
    properties.add("lastName");
    properties.add("addresses.type");
    properties.add("addresses.state");
    return properties;
}
```

1.5.5.1.5 Deleting Contacts

When deleting a contact record, use the ContactRepository deleteContact method. Identify the target record to delete by its unique key.

Code Snippet

```
//for this example, it is assumed that you already know the target contact record's unique key (where uniqueKey is a String variable)
```

```
protected ContactRepository contactRepository;
contactRepository.deleteContact(uniqueKey);
```

1.5.5.2 Accounts

The following .JAVA code samples show how you can use TeamConnect® Enterprise Web Services for functions specific to account records.

1.5.5.2.1 Creating accounts

The following sample creates an account record. In the sample, required fields are commented. During account creation, it is optional to set posting criteria or definitions on which types of financial records can post to the account (for example, tasks, expenses, invoices).

Code Snippet

```
Date date = new Date();
protected AccountRepository accountRepository;
AccountCreate account = createAccount();
String accountUniqueKey = accountRepository.insertAccount(account);
private String createAccount() throws Exception {
    AccountCreate account = new AccountCreate();
    //name is a required field
    account.setName("2009-Q1");
    //startOn date is a required field
    account.setStartOn(01-JAN-09);
    //endOn date is a required field
    account.setEndOn(31-MAR-09);
    //allocationLimit is a required field
    account.setAllocationLimit(25000.00);
    //(optional) set the account type as BUDGET or RESERVE
    account.setType(AccountType.BUDGET);
    //(optional) set the account overdraft type
    account.setAccountOverdraftType(AccountOverdraftType.ALLOW_NEGATIVE);
    //(optional) if this is a child account, set the parent account
    account.setParentAccountUniqueKey("ACCT_4141");
    returns account;
}
```

Code Snippet for setting account posting criteria

```
//the following sample code would be used as an excerpt from the above sample for
creating an account; it is presented separately just to simplify the process for
setting account posting criteria
//the following section sets posting criteria that allows tasks of a certain
category to post to this account
account.setAllowTask(true);
account.setTaskCategoryUniqueKey(taskCategoryUniqueKey);
```

```
account.setTaskPercent(100);
//the following section sets posting criteria that allows expenses of a certain
category to post to this account
account.setAllowExpense(true);
account.setExpenseCategoryUniqueKey(expenseCategoryUniqueKey);
account.setExpensePercent(100);
//the following section sets posting criteria that allows invoice expense line
items of a certain category to post to this account
account.setAllowInvoiceExpense(true);
account.setInvoiceExpenseCategoryUniqueKey(invoiceExpCategoryUniqueKey);
account.setInvoiceExpensePercent(100);
//the following section sets posting criteria that allows invoice fee line items of
a certain category to post to this account
account.setAllowInvoiceTask(true);
account.setInvoiceTaskCategoryUniqueKey(invoiceTaskCategoryUniqueKey);
account.setInvoiceTaskPercent(100);
```

1.5.5.2.2 Updating accounts

The following sample updates an existing account record.

Code Snippet

```
Date date = new Date();
protected AccountRepository accountRepository;
//for this example, it is assumed that you already know the target account record's
unique key (where uniqueKey is a String variable)
AccountUpdate account = new AccountUpdate();
account.setUniqueKey(uniqueKey);
account.setName("Budget 2009 Q2");
account.setAllocated(15000.00);
account.setAllocationLimit(20000.00);
account.setUsed(5000.00);
account.setAvailable(10000.00);
account.setStartOn(date);
account.setAccountOverdraftType(AccountOverdraftType.ALLOW_NEGATIVE);
account.setType(AccountType.BUDGET);
account.setActive(true);
//setting the autoPost property to true indicates that financial transactions can
post to this account;
account.setAutoPost(true);
account.setParentAccountUniqueKey("ACCT_1519");
accountRepository.updateAccount(account);
```

1.5.5.2.3 Reading accounts

The following sample reads an existing account record.

Code Snippet

```
//for this example, it is assumed that you already know the target account record's
unique key (where uniqueKey is a String variable)
protected AccountRepository accountRepository;
public void readAccount() throws Exception {
    Account account = (Account) accountRepository.readAccount(uniqueKey,
    getPropertiesToRead());
}
private List<String> getPropertiesToRead() {
    List<String> properties = new List<String>();
    properties.add("createdBy");
    properties.add("createdOn");
    properties.add("modifiedBy");
    properties.add("modifiedOn");
    properties.add("name");
    properties.add("allocated");
    properties.add("allocationLimit");
    properties.add("used");
    properties.add("available");
    properties.add("startOn");
    properties.add("endOn");
    properties.add("expensePercent");
    properties.add("invoiceExpensePercent");
    properties.add("invoiceTaskPercent");
    properties.add("taskPercent");
    properties.add("allowExpense");
    properties.add("allowTask");
    properties.add("allowInvoiceExpense");
    properties.add("allowInvoiceTask");
    properties.add("accountInvolvedType");
    properties.add("accountVendorType");
    properties.add("accountProjectType");
    properties.add("accountOverdraftType");
    properties.add("type");
    properties.add("active");
    properties.add("autoPost");
    properties.add("categories");
    properties.add("expenseCategory");
    properties.add("taskCategory");
    properties.add("projectCategory");
    properties.add("invoiceExpenseCategory");
```

```
    properties.add("invoiceTaskCategory");
    properties.add("involved");
    properties.add("project");
    properties.add("vendor");
    properties.add("parentAccount");
    return properties;
}
```

1.5.5.2.4 Reading child accounts

The following sample reads child accounts of a parent account record. Identify the parent account by its unique key. In the sample, a complete list of account properties is provided but in your program, you can remove unnecessary properties to improve performance. Only the listed property values will be returned with the child accounts in the search results.

Code Snippet

```
//for this example, it is assumed that you already know the target parent account
record's unique key (where uniqueKey is a String variable)
protected AccountRepository accountRepository;
public void readChildAccounts() throws Exception {
    List<Account> accounts = (List<Account>)
    accountRepository.readChildAccounts(uniqueKey, getPropertiesToRead());
}
private List<String> getPropertiesToRead() {
    List<String> properties = new List<String>();
    properties.add("createdBy");
    properties.add("createdOn");
    properties.add("modifiedBy");
    properties.add("modifiedOn");
    properties.add("name");
    properties.add("allocated");
    properties.add("allocationLimit");
    properties.add("used");
    properties.add("available");
    properties.add("startOn");
    properties.add("endOn");
    properties.add("expensePercent");
    properties.add("invoiceExpensePercent");
    properties.add("invoiceTaskPercent");
    properties.add("taskPercent");
    properties.add("allowExpense");
    properties.add("allowTask");
    properties.add("allowInvoiceExpense");
    properties.add("allowInvoiceTask");
    properties.add("accountInvolvedType");
    properties.add("accountVendorType");
```

```
    properties.add("accountProjectType");
    properties.add("accountOverdraftType");
    properties.add("type");
    properties.add("active");
    properties.add("autoPost");
    properties.add("categories");
    properties.add("expenseCategory");
    properties.add("taskCategory");
    properties.add("projectCategory");
    properties.add("invoiceExpenseCategory");
    properties.add("invoiceTaskCategory");
    properties.add("involved");
    properties.add("project");
    properties.add("vendor");
    properties.add("parentAccount");
    return properties;
}
```

1.5.5.2.5 Searching accounts

The following sample searches for account records where the name contains "2009".

Code Snippet

```
//for this example, it is assumed that at least one account record that meets the
search criterion already exists
protected AccountRepository accountRepository;
public void searchAccounts() throws Exception {
    StringFieldCriterion fieldCriterion = new StringFieldCriterion();
    fieldCriterion.setComparator(StringComparator.CONTAINS);
    LegacySearchFieldPathExpression fieldPathExpression = new
    LegacySearchFieldPathExpression();
    fieldPathExpression.setSearchKeyPath("name");
    fieldCriterion.setFieldPath(fieldPathExpression);
    fieldcriterion.getValue().add("2009");
    FieldSearchClause searchCriteria = new FieldSearchClause();
    searchCriteria.setOperator(LogicOperator.AND);
    searchCriteria.getCriteria().add(fieldCriterion);
    List<Account> accounts =
    accountRepository.readAccountsByCriteria(searchCriteria, 100,
    getPropertiesToRead());
}
private List<String> getPropertiesToRead() {
    List<String> properties = new List<String>();
    properties.add("createdBy");
    properties.add("createdOn");
    properties.add("modifiedBy");
```

```
        properties.add("modifiedOn");
        properties.add("name");
        properties.add("allocated");
        properties.add("allocationLimit");
        properties.add("used");
        properties.add("available");
        properties.add("startOn");
        properties.add("endOn");
        properties.add("expensePercent");
        properties.add("invoiceExpensePercent");
        properties.add("invoiceTaskPercent");
        properties.add("taskPercent");
        properties.add("allowExpense");
        properties.add("allowTask");
        properties.add("allowInvoiceExpense");
        properties.add("allowInvoiceTask");
        properties.add("accountInvolvedType");
        properties.add("accountVendorType");
        properties.add("accountProjectType");
        properties.add("accountOverdraftType");
        properties.add("type");
        properties.add("active");
        properties.add("autoPost");
        properties.add("categories");
        properties.add("expenseCategory");
        properties.add("taskCategory");
        properties.add("projectCategory");
        properties.add("invoiceExpenseCategory");
        properties.add("invoiceTaskCategory");
        properties.add("involved");
        properties.add("project");
        properties.add("vendor");
        properties.add("parentAccount");
        return properties;
    }
```

1.5.5.2.6 Activating accounts

The following sample activates an existing account record.

Code Snippet

```
// this sample assumes an account already exists with the unique key (String) value
equal to uniqueKey
protected AccountRepository accountRepository;
accountRepository.activateAccount(uniqueKey);
```

1.5.5.2.7 Deactivating accounts

The following sample deactivates an existing active account record.

Code Snippet

```
// this sample assumes an active account already exists with the unique key
(String) value equal to uniqueKey
protected AccountRepository accountRepository;
accountRepository.deactivateAccount(uniqueKey);
```

1.5.5.2.8 Allocating money to accounts

The following sample allocates money to an existing active account record.

Code Snippet

```
// this sample assumes an active account already exists with the unique key
(String) value equal to uniqueKey
protected AccountRepository accountRepository;
//the allocated variable defines the monetary amount to allocate to the account
BigDecimal allocated = new BigDecimal("1000.000000000000");
//the third parameter of the allocateMoney method is a String description of the
fund allocation transaction
accountRepository.allocateMoney(uniqueKey, allocated, "test money allocation
through Web Services");
```

1.5.5.2.9 Transferring money between accounts

The following sample transfers funds from a parent account to one of its child accounts. The transferMoney method can also transfer funds from a child account to its parent account. For the sample below, it is assumed that the following already exist:

- a parent account which is active and has money already allocated to it, where the record's unique key is a String equal to parentUniqueKey
- a child account for the parent which is active, where the record's unique key is a String equal to childUniqueKey

Code Snippet

```
protected AccountRepository accountRepository;
//the third parameter is the monetary amount to transfer between the parent and
child account; the fourth parameter is a String description of the transfer
transaction
accountRepository.transferMoney(parentUniqueKey, childUniqueKey, new
BigDecimal("3333.00"), "money transferred through Web Services");
```

1.5.5.2.10 Withdrawing money from accounts

The following sample withdraws funds from an account. For the sample, it is assumed that an active account already exists.

Code Snippet

```
// this sample assumes an active account already exists with the unique key
(String) value equal to uniqueKey
protected AccountRepository accountRepository;
// Withdraw money from the account
BigDecimal used = new BigDecimal("100.000000000000");
//the third parameter is a String description of the withdrawal transaction
accountRepository.withdrawMoney(uniqueKey, used, "test allocating");
```

1.5.5.2.11 Deleting accounts

The following sample deletes an existing account record.

Code Snippet

```
//for this example, it is assumed that you already know the target account record's
unique key (where uniqueKey is a String variable)
protected AccountRepository accountRepository;
accountRepository.deleteAccount(uniqueKey);
```

1.5.5.3 Appointments

The following .JAVA code samples show how you can use TeamConnect® Enterprise Web Services for functions related to appointment records.

1.5.5.3.1 Creating appointments

The following sample creates an appointment record.

Code Snippet

```
protected AppointmentRepository appointmentRepository;
private String createAppointment() throws Exception {
    Date date = new Date();
    AppointmentCreate appointment = new AppointmentCreate();
    appointment.setSubject("Quarterly Meeting - Q2 2009");
    appointment.setLocation("Build A-5");
    appointment.setAssignedOn(date);
    appointment.setStartOn(date);
    appointment.setEndOn(date);
    appointment.setAllDay(true);
    appointment.setProjectUniqueKey("DISP_0003");
    appointment.getAttendeeCreates().add(attendees);
```

```
    appointment.getResourceCreates().add(resources);
    appointment.getCategories().add(categories);
    return appointmentRepository.insertAppointment(appointment);
}
private List<AttendeeCreate> attendeeCreate() {
    List<AttendeeCreate> attendees = new List<AttendeeCreate>();
    AttendeeCreate attendee1 = new AttendeeCreate();
    attendee1.setUserUniqueKey("USER_1005");
    attendee1.setAttendanceType(AttendanceType.WILL_ATTEND);
    AttendeeCreate attendee2 = new AttendeeCreate();
    attendee2.setUserUniqueKey("USER_1013");
    attendee2.setAttendanceType(AttendanceType.TENTATIVE);
    attendees.add(attendee1);
    attendees.add(attendee2);
    return attendees;
}
private List<AppointmentResourceCreate> appointmentResourceCreate()
{
    List<AppointmentResourceCreate> appointmentResources = new
    List<AppointmentResourceCreate>();
    AppointmentResourceCreate apptResource = new AppointmentResourceCreate();
    //for this sample, it is assumed that for the Resource Type system lookup
    table, there is a table item with the unique key (or Tree position),
    "CRMA" (for Conference Room A)
    apptResource.setTypeUniqueKey("CRMA");
    appointmentResources.add(apptResource);
    return appointmentResources;
}
private List<Category> categoryAdd() {
    List<Category> categories = new ListArray<Category>();
    Category cat = new Category();
    //the next two lines identify a category and then add the category to the
    appointment record
    cat.setUniqueKey("APPT_TRIA");
    categories.add(cat);
    return categories;
}
```

1.5.5.3.2 Updating appointments

The following sample updates an appointment record with given unique key.

Code Snippet

```
//for this sample, it is assumed that an appointment record already exists with a
unique key String equal to uniqueKey
protected AppointmentRepository appointmentRepository;
```

```
public void test_updateAppointment() throws Exception {
    Date date = new Date();
    AppointmentUpdate appointment = new AppointmentUpdate();
    //use the setUniqueKey method to identify the target task record to update by
    its unique key
    appointment.setUserUniqueKey("user_1003");
    appointment.setAssignedOn(date);
    appointment.setStartOn(date);
    appointment.setEndOn(date);
    appointment.setAllDay(true);
    appointment.setProjectUniqueKey("DISP_0003");
    appointment.getAttendeeUpdates().add(attendees);
    appointmentRepository.updateAppointment(appointment);
}
private List<AttendeeUpdate> attendeeUpdate() {
    List<AttendeeUpdate> attendees = new List<AttendeeUpdate>();
    AttendeeUpdate attendee1 = new AttendeeUpdate();
    //identify which attendee to update by the record's unique key
    attendee1.setUniqueKey("ATTN_1001");
    attendee1.setUserUniqueKey("USER_1005");
    attendee1.setAttendanceType(AttendanceType.WILL_NOT_ATTEND);
    attendees.add(attendee1);
    return attendees;
}
```

1.5.5.3.3 Reading appointments

The following sample reads an appointment record and returns the specified property values.

Code Snippet

```
protected AppointmentRepository appointmentRepository;
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    //the list of all available properties displays below but to increase
    efficiency in your searches, you can omit unnecessary properties from your
    application and those values will not be returned
    props.add("uniqueKey");
    props.add("version");
    props.add("subject");
    props.add("categories");
    props.add("createdOn");
    props.add("createdBy");
    props.add("modifiedBy");
    props.add("modifiedOn");
    props.add("location");
```

```
    props.add("startOn");
    props.add("endOn");
    props.add("allDay");
    props.add("project");
    props.add("attendees");
    props.add("resources");
    props.add("categories");
    return props;
}
private Appointment readAppointment() throws Exception {
    Appointment readAppointment = appointmentRepository.readAppointment(uniqueKey,
    getPropertiesToRead());
    return readAppointment;
}
```

1.5.5.3.4 Searching appointments

The following sample searches for one or more appointment records and returns the specified property values. For this sample, it is assumed there is one or more appointments already existing with a subject String field value equal to subject.

Code Snippet

```
protected AppointmentRepository appointmentRepository;
private List<Appointment> test_readAppointmentsByCriteria() throws
Exception {
    // Criterion for appointment searching
    StringFieldCriterion fieldCriterion = new StringFieldCriterion();
    fieldCriterion.setComparator(StringComparator.EQUALS_ENFORCE_CASE);
    LegacySearchFieldPathExpression fieldPathExpression = new
    LegacySearchFieldPathExpression();
    fieldPathExpression.setSearchKeyPath("subject");
    fieldCriterion.setFieldPath(fieldPathExpression);
    fieldCriterion.getValue().add("Client meeting");
    FieldSearchClause searchCriteria = new FieldSearchClause();
    searchCriteria.setOperator(LogicOperator.AND);
    searchCriteria.getCriteria().add(fieldCriterion);
    appointments =
    appointmentRepository.readAappointmentsByCriteria(searchCriteria, 100,
    getPropertiesToRead());
}
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    //the list of all available properties displays below but to increase
    efficiency in your searches, you can omit unnecessary properties from your
    application and those values will not be returned
    props.add("uniqueKey");
    props.add("version");
```

```
    props.add("subject");
    props.add("categories");
    props.add("createdOn");
    props.add("createdBy");
    props.add("modifiedBy");
    props.add("modifiedOn");
    props.add("location");
    props.add("startOn");
    props.add("endOn");
    props.add("allDay");
    props.add("project");
    props.add("attendees");
    props.add("resources");
    props.add("categories");
    return props;
}
```

1.5.5.3.5 Deleting appointments

The following sample deletes an existing appointment.

Code Snippet

```
//for this sample it is assumed that a appointment record exists with a unique key
String equal to uniqueKey
protected AppointmentRepository appointmentRepository;
public void test_deleteAppointment() throws Exception {
    appointmentRepository.deleteAppointment(uniqueKey);
}
```

1.5.5.4 Document Records

The following .JAVA code samples show how you can use TeamConnect® Enterprise Web Services for functions related to document records.

1.5.5.4.1 Creating document records

There are four types of document records: file, folder, hyperlink, shortcut. The following samples create a file. Separate ContactRepository methods are provided to create folders and shortcuts.

Code Snippet for creating a document file

```
Date date = new Date();
protected DocumentRepository documentRepository;
String uniqueKey = documentRepository.insertDocument(document);
private void createDocument() {
    DocumentCreate document = new DocumentCreate();
    document.setName("test_document.txt");
```

```
    document.setContent("test document content".getBytes());
    document.setType(DocumentType.FILE);
    document.setParentFolderUniqueKey("DOCU_11");
    document.setContentTypeUniqueKey("Text");
    return document;
}
```

1.5.5.4.2 Updating documents

The following samples show how to update documents for the available types (file, folder, hyperlink, shortcut).

Code Snippet for updating a document file

```
Date date = new Date();
protected DocumentRepository documentRepository;
documentRepository.updateDocument(document);
private void updateDocument() {
    DocumentUpdate document = new DocumentUpdate();
    document.setUniqueKey("DOCU_2010");
    document.setContent(byte[] value);
    document.setName("receipt012109");
    return document;
}
```

Code Snippet for updating a document folder

```
Date date = new Date();
protected DocumentRepository documentRepository;
documentRepository.updateDocument(document);
private void updateDocument() {
    DocumentUpdate document = new DocumentUpdate();
    document.setUniqueKey("DOCU_2011");
    document.setName("Correspondence_Jan09");
    document.setParentFolderUniqueKey("DOCU_3005");
    return document;
}
```

Code Snippet for updating a document hyperlink

```
Date date = new Date();
protected DocumentRepository documentRepository;
documentRepository.updateDocument(document);
private void updateDocument() {
    DocumentUpdate document = new DocumentUpdate();
    document.setUniqueKey("DOCU_2011");
    document.setName("BusinessWeek_URL");
    document.setUrl("http://businessweek.com");
```

```
    return document;
}
```

Code Snippet for updating a document shortcut

```
Date date = new Date();
protected DocumentRepository documentRepository;
documentRepository.updateDocument(document);
private void updateDocument() {
    DocumentUpdate document = new DocumentUpdate();
    document.setUniqueKey("DOCU_2010");
    document.setName("shortcutToWorksheet");
    return document;
}
```

1.5.5.4.3 Reading documents

The following samples show how to read documents. Note that certain properties you can request to be returned are only relevant to particular document types (for example, the url property is only relevant to the document type, hyperlink). Comments are provided for properties that are document type-specific.

Code Snippet

```
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    //the list of all available properties displays below but to increase
    efficiency in your searches, you can omit unnecessary properties from your
    application and those values will not be returned
    props.add("version");
    props.add("subject");
    props.add("categories");
    props.add("createdOn");
    props.add("createdBy");
    props.add("modifiedBy");
    props.add("modifiedOn");
    props.add("name");
    props.add("author");
    props.add("authorDate");
    props.add("size");
    //the contentType is only relevant to a document of type FILE
    props.add("contentType");
    //the contentType is only relevant to a document of type FILE
    props.add("content");
    props.add("type");
    props.add("parentFolder");
    props.add("checkedOutBy");
    props.add("checkedOutOn");
```

```
    props.add("checkedInBy");
    props.add("checkedInOn");
    props.add("versionText");
    //the url is only relevant to a document of type HYPERLINK
    props.add("url");
    //the contentType is only relevant to a document of type SHORTCUT
    props.add("shortcutToDocument");
    return props;
}
private Document readDocument() throws Exception {
    Document readDocument = documentRepository.readDocument(uniqueKey,
    getPropertiesToRead());
    return readDocument;
}
```

1.5.5.4.4 Reading child documents for a document

Performs one of the following:

- Gets child documents (files) for a given document folder.
- Gets previous versions of a document (files) for a given document file.

For both operations, documents are returned with only specified property values. For a list of available properties, see the type class, Document.

Code Snippet

```
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    //the list of all available properties displays below but to increase
    efficiency in your searches, you can omit unnecessary properties from your
    application and those values will not be returned
    props.add("version");
    props.add("subject");
    props.add("categories");
    props.add("createdOn");
    props.add("createdBy");
    props.add("modifiedBy");
    props.add("modifiedOn");
    props.add("name");
    props.add("author");
    props.add("authorDate");
    props.add("size");
    //the contentType is only relevant to a document of type FILE
    props.add("contentType");
    //the contentType is only relevant to a document of type FILE
    props.add("content");
```

```
    props.add("type");
    props.add("parentFolder");
    props.add("checkedOutBy");
    props.add("checkedOutOn");
    props.add("checkedInBy");
    props.add("checkedInOn");
    props.add("versionText");
    //the url is only relevant to a document of type HYPERLINK
    props.add("url");
    //the contentType is only relevant to a document of type SHORTCUT
    props.add("shortcutToDocument");
    return props;
}
private Document readChildDocument() throws Exception {
    //the first parameter, parentUniqueKey, identifies the unique key of the parent
    folder whose child documents' properties to return; where child documents would
    include files, subfolders, hyperlinks, and shortcuts included in the parent
    folder recursively
    List<Document> readChildDocument =
    documentRepository.readChildDocuments("DOCU_2534",
    getPropertiesToRead());
    return readChildDocument;
}
```

1.5.5.4.5 Reading documents by path

Retrieves a Document from the repository based on a folder path.

Code Snippet

```
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    //the list of all available properties displays below but to increase
    efficiency in your searches, you can omit unnecessary properties from your
    application and those values will not be returned
    props.add("version");
    props.add("subject");
    props.add("categories");
    props.add("createdOn");
    props.add("createdBy");
    props.add("modifiedBy");
    props.add("modifiedOn");
    props.add("name");
    props.add("author");
    props.add("authorDate");
    props.add("size");
    //the contentType is only relevant to a document of type FILE
```

```
    props.add("contentType");
    //the contentType is only relevant to a document of type FILE
    props.add("content");
    props.add("type");
    props.add("parentFolder");
    props.add("checkedOutBy");
    props.add("checkedOutOn");
    props.add("checkedInBy");
    props.add("checkedInOn");
    props.add("versionText");
    //the url is only relevant to a document of type HYPERLINK
    props.add("url");
    //the contentType is only relevant to a document of type SHORTCUT
    props.add("shortcutToDocument");
    return props;
}
private Document readDocumentByPath() throws Exception {
    //the first parameter, path, identifies the document folder path to
    the target document whose specified properties to return
    Document document =
    documentRepository.readDocumentByPath("TopLevel_Users_john",
    getPropertiesToRead());
    return document;
}
```

1.5.5.4.6 Reading the child document of a parent folder

Retrieves a document from the repository given the file name and the unique ID of its parent folder.

Code Snippet

```
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    //the list of all available properties displays below but to increase
    efficiency in your searches, you can omit unnecessary properties from your
    application and those values will not be returned
    props.add("version");
    props.add("subject");
    props.add("categories");
    props.add("createdOn");
    props.add("createdBy");
    props.add("modifiedBy");
    props.add("modifiedOn");
    props.add("name");
    props.add("author");
    props.add("authorDate");
```

```
    props.add("size");
    //the contentType is only relevant to a document of type FILE
    props.add("contentType");
    //the contentType is only relevant to a document of type FILE
    props.add("content");
    props.add("type");
    props.add("parentFolder");
    props.add("checkedOutBy");
    props.add("checkedOutOn");
    props.add("checkedInBy");
    props.add("checkedInOn");
    props.add("versionText");
    //the url is only relevant to a document of type HYPERLINK
    props.add("url");
    //the contentType is only relevant to a document of type SHORTCUT
    props.add("shortcutToDocument");
    return props;
}
private Document readChildDocumentForName() throws Exception {
    //the first parameter, parentUniqueKey, identifies the unique key of the parent
    folder whose child documents' properties to return; where child documents would
    include files, subfolders, hyperlinks, and shortcuts included in the parent
    folder recursively
    //the second parameter identifies the target document file name, including file
    extension
    List<Document> readChildDocumentForName =
    documentRepository.readChildDocuments("DOCU_2789",
    "invoice101009_4009_attachment.doc", getPropertiesToRead());
    return readChildDocument;
}
```

1.5.5.4.7 Reading the Document Folder for a User

Retrieves a document folder from the repository given the unique key of the folder's owner (user).

Code Snippet

```
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    //the list of all available properties displays below but to increase
    efficiency in your searches, you can omit unnecessary properties from your
    application and those values will not be returned
    props.add("version");
    props.add("subject");
    props.add("categories");
    props.add("createdOn");
    props.add("createdBy");
    props.add("modifiedBy");
```

```
    props.add("modifiedOn");
    props.add("name");
    props.add("author");
    props.add("authorDate");
    props.add("size");
    props.add("content");
    props.add("type");
    props.add("parentFolder");
    props.add("checkedOutBy");
    props.add("checkedOutOn");
    props.add("checkedInBy");
    props.add("checkedInOn");
    props.add("versionText");
    return props;
}
private Document readDocumentFolderForDocumentOwner() throws Exception {
    //the first parameter, uniqueKey, identifies the unique key of the user whose
    TeamConnect Documents area user folder to return (for example, the path to the
    user folder for Al would look like Top Level\Users\al)
    document readDocumentFolderForDocumentOwner =
    documentRepository.readDocumentFolderForDocumentOwner("USER_1223",
    getPropertiesToRead());
    return readUserFolder;
}
```

1.5.5.4.8 Searching documents

The following samples show how to search documents by name, parent folder, date last modified, and author.

Code Snippet

```
\\for this sample it is assumed that a Date variable, date, exists representing a
past date
protected DocumentRepository documentRepository;
StringFieldCriterion fieldCriterion1 = new StringFieldCriterion();
fieldCriterion1.setComparator(StringComparator.CONTAINS);
LegacySearchFieldPathExpression fieldPathExpression1 = new
LegacySearchFieldPathExpression();
fieldPathExpression1.setSearchKeyPath("name");
fieldCriterion1.setFieldPath(fieldPathExpression);
fieldCriterion1.getValue().add("Invoice_10508");
StringFieldCriterion fieldCriterion2 = new StringFieldCriterion();
fieldCriterion2.setComparator(StringComparator.EQUALS);
LegacySearchFieldPathExpression fieldPathExpression2 = new
LegacySearchFieldPathExpression();
fieldPathExpression2.setSearchKeyPath("parentFolder.uniqueKey");
fieldCriterion2.setFieldPath(fieldPathExpression);
```

```
fieldCriterion2.getValue().add("DOCU_1099");
DateFieldCriterion fieldCriterion3 = new DateFieldCriterion();
fieldCriterion3.setComparator(DateComparator.EQUALS);
LegacySearchFieldPathExpression fieldPathExpression3 = new
LegacySearchFieldPathExpression();
fieldPathExpression3.setSearchKeyPath("modifiedOn");
fieldCriterion3.setFieldPath(fieldPathExpression);
fieldCriterion3.getValue().add(date);
StringFieldCriterion fieldCriterion4 = new StringFieldCriterion();
fieldCriterion4.setComparator(StringComparator.EQUALS);
LegacySearchFieldPathExpression fieldPathExpression4 = new
LegacySearchFieldPathExpression();
fieldPathExpression4.setSearchKeyPath("author.uniqueKey");
fieldCriterion4.setFieldPath(fieldPathExpression);
fieldCriterion4.getValue().add("CONT_1015");
FieldSearchClause searchCriteria = new FieldSearchClause();
searchCriteria.setOperator(LogicOperator.AND);
searchCriteria.getCriteria().add(fieldCriterion1);
searchCriteria.getCriteria().add(fieldCriterion2);
searchCriteria.getCriteria().add(fieldCriterion3);
searchCriteria.getCriteria().add(fieldCriterion4);
List<Document> searchDocument =
documentRepository.readDocumentsByCriteria(searchCriteria, 100,
getPropertiesToRead());
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    //the list of all available properties displays below but to increase
    efficiency in your searches, you can omit unnecessary properties from your
    application and those values will not be returned
    props.add("version");
    props.add("subject");
    props.add("categories");
    props.add("createdOn");
    props.add("createdBy");
    props.add("modifiedBy");
    props.add("modifiedOn");
    props.add("name");
    props.add("author");
    props.add("authorDate");
    props.add("size");
    //the contentType is only relevant to a document of type FILE
    props.add("contentType");
    //the contentType is only relevant to a document of type FILE
    props.add("content");
    props.add("type");
    props.add("parentFolder");
```

```
        props.add("checkedOutBy");
        props.add("checkedOutOn");
        props.add("checkedInBy");
        props.add("checkedInOn");
        props.add("versionText");
        //the url is only relevant to a document of type HYPERLINK
        props.add("url");
        //the contentType is only relevant to a document of type SHORTCUT
        props.add("shortcutToDocument");
        return props;
    }
```

1.5.5.4.9 Creating document shortcuts

Creates a shortcut or link to a document (file, folder, or existing shortcut) with a given unique key. The parentFolderUniqueKey defines the folder where the shortcut will be created. The unique key of the resulting shortcut (type of document record) is returned.

Code Snippet

```
protected DocumentRepository documentRepository;
//the first parameter, documentUniqueKey, identifies an existing document to create a link to
//the second parameter, parentFolderUniqueKey, identifies the folder where the shortcut will be created
String uniqueKey = documentRepository.createShortcut("DOCU_1451", "DOCU_1771");
```

1.5.5.4.10 Creating subfolders

Creates a sub-folder with given name in given parent folder. The unique key of the resulting folder (type of document record) is returned.

Code Snippet

```
protected DocumentRepository documentRepository;
//the first parameter, name, sets the subfolder name
//the second parameter, parentUniqueKey, identifies the folder where the folder will be created
String uniqueKey = documentRepository.createSubFolder("Child Folder", "DOCU_1010");
```

1.5.5.4.11 Creating hyperlinks with the default Document category

Creates a hyperlink and sets its category to the default Document category.

Code Snippet

```
protected DocumentRepository documentRepository;
//the first parameter, parentFolderUniqueKey, identifies the folder where the hyperlink will be created name, sets the subfolder name
//the second parameter, url, sets the website URL target for the hyperlink
```

```
//the third parameter, name, sets the hyperlink name that displays in the
application UI
//the fourth parameter, contactUniqueKey, identifies the hyperlink record author
String uniqueKey = documentRepository.createHyperlinkDefaultCategory("DOCU_1753",
"http://www.mitratech.com", "Mitratech URL", "CONT_1005");
```

1.5.5.4.12 Creating hyperlinks

Creates a hyperlink.

Code Snippet

```
protected DocumentRepository documentRepository;
//the first parameter, parentFolderUniqueKey, identifies the folder where the
hyperlink will be created name, sets the subfolder name
//the second parameter, url, sets the website URL target for the hyperlink
//the third parameter, name, sets the hyperlink name that displays in the
application UI
//the fourth parameter, documentCategoryUniqueKey, identifies the document category
to set as the hyperlink category
//the fifth parameter, contactUniqueKey, identifies the hyperlink record author
String uniqueKey = documentRepository.createHyperlinkDefaultCategory("DOCU_1753",
"http://www.mitratech.com", "Mitratech URL", "DOCU_1855", "CONT_1005");
```

1.5.5.4.13 Copying documents

Copies a document to a new parent folder.

Code Snippet

```
protected DocumentRepository documentRepository;
//the first parameter, documentUniqueKey, identifies an existing document to copy
//the second parameter, parentFolderUniqueKey, identifies the folder where the
document copy will be created
documentRepository.copyDocument("DOCU_1700", "DOCU_1811");
```

1.5.5.4.14 Moving documents

Moves a document to a new parent folder.

Code Snippet

```
protected DocumentRepository documentRepository;
//the first parameter, documentUniqueKey, identifies an existing document to move
//the second parameter, newParentFolderUniqueKey, identifies the folder where the
document will be moved
documentRepository.moveDocument("DOCU_1900", "DOCU_1024");
```

1.5.5.4.15 Checking out documents

Attempts to check out a document with a given unique key.

Code Snippet

```
protected DocumentRepository documentRepository;
//the first parameter, documentUniqueKey, identifies an existing document to check out
documentRepository.checkOut("DOCU_2900");
```

1.5.5.4.16 Undoing a document checkout

Cancels a checkout operation on a document with a given unique key.

Code Snippet

```
protected DocumentRepository documentRepository;
//the parameter, documentUniqueKey, identifies an existing document to undo the checkout operation for
documentRepository.undoCheckOut("DOCU_2900");
```

1.5.5.4.17 Checking in documents

Attempts to check in a document (of type file) with a given unique key.

Code Snippet

```
byte[] docContent = "test document content".getBytes();
protected DocumentRepository documentRepository;
//the first parameter, documentUniqueKey, identifies an existing document to check in a new version for
//the second parameter, content, contains the updated file content to check in
//the third parameter, version, identifies the numeric version of the file you are checking in
documentRepository.checkIn("DOCU_2901", docContent, "1.3");
```

1.5.5.4.18 Reverting a document to a previous version

Attempts to revert a given document to a given old version.

Code Snippet

```
protected DocumentRepository documentRepository;
//the first parameter, documentUniqueKey, identifies the version of the document file (for example the current version) to replace
//the second parameter, oldKey, identifies the version of the document file to revert to (sets this file to the current version)
documentRepository.revert("DOCU_2902", "DOCU_2892");
```

1.5.5.4.19 Deleting documents

The following sample deletes a document. You must get the unique key value for record to delete.

Code Snippet

```
//for this sample it is assumed that a document exists with the unique key equal to
a String, uniqueKey
protected DocumentRepository documentRepository;
documentRepository.deleteDocument(uniqueKey);
```

1.5.5.5 Expenses

The following .JAVA code samples show how you can use TeamConnect® Enterprise Web Services for functions related to expense records.

1.5.5.5.1 Creating expenses

The following sample creates an expense record.

Code Snippet

```
protected ExpenseRepository expenseRepository;
private String createExpense() throws Exception {
    Date date = new Date();
    ExpenseCreate expense = new ExpenseCreate();
    expense.setContactUniqueKey("CONT_1003");
    //expenseDate is a required property
    expense.setExpenseDate(date);
    //expensedByUniqueKey is a required property
    expense.setExpensedByUniqueKey("EXPE_1212");
    //shortDescription is a required property
    expense.setShortDescription(String value);
    expense.setProjectUniqueKey(String value);
    expense.setQuantity(BigDecimal value);
    expense.setUnitPrice(BigDecimal value);
    return expenseRepository.insertExpense(expense);
}
```

1.5.5.5.2 Updating expenses

The following sample updates an expense record's expenseDate, projectUniqueKey, quantity, totalAmount, unitPrice values.

Code Snippet

```
//for this sample, it is assumed that an expense record already
exists with a unique key String equal to uniqueKey
protected ExpenseRepository expenseRepository;
```

```
public void test_updateExpense() throws Exception {
    Date date = new Date();
    ExpenseUpdate expenseUpdate = new ExpenseUpdate();
    //use the setUniqueKey method to identify the target expense record to update
    by its unique key
    expenseUpdate.setUniqueKey(uniqueKey);
    expenseUpdate.setExpenseDate(date);
    expenseUpdate.setProjectUniqueKey("DISP_2003");
    expenseUpdate.setQuantity(15.0);
    expenseUpdate.setUnitPrice(2.0);
    //Note that the totalAmount property is calculated by the product of the
    quantity and unitPrice property values. In the expense update sample provided,
    the totalAmount property will automatically be updated to 30.0
    expenseRepository.updateExpense(expenseUpdate);
}
```

1.5.5.5.3 Reading expenses

The following sample reads an expense record and returns the specified property values.

Code Snippet

```
protected ExpenseRepository expenseRepository;
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    //the list of all available properties displays below but to ncrease efficiency
    in your searches, you can omit unnecessary properties from your application and
    those values will not be returned
    props.add("shortDescription");
    props.add("categories");
    props.add("createdOn");
    props.add("modifiedBy");
    props.add("createdBy");
    props.add("modifiedOn");
    props.add("version");
    props.add("contact");
    props.add("expenseDate");
    props.add("expensedBy");
    props.add("project");
    props.add("quantity");
    props.add("unitPrice");
    props.add("totalAmount");
    return props;
}
private Expense readExpense() throws Exception {
    Expense readExpense = expenseRepository.readExpense(uniqueKey,
    getPropertiesToRead());
```

```
    return readExpense;
}
```

1.5.5.5.4 Searching expenses

The following sample searches for one or more expense records and returns the specified property values. For this sample, it is assumed there are one or more expenses already existing with a subject String field value equal to subject.

Code Snippet

```
protected ExpenseRepository expenseRepository;
private List<Expense> test_readExpensesByCriteria() throws Exception
{
    // Criterion for expense searching
    StringFieldCriterion fieldCriterion = new StringFieldCriterion();
    fieldCriterion.setComparator(StringComparator.EQUALS_ENFORCE_CASE);
    LegacySearchFieldPathExpression fieldPathExpression = new
    LegacySearchFieldPathExpression();
    fieldPathExpression.setSearchKeyPath("shortDescription");
    fieldCriterion.setFieldPath(fieldPathExpression);
    fieldCriterion.getValue().add(subject);
    FieldSearchClause searchCriteria = new FieldSearchClause();
    searchCriteria.setOperator(LogicOperator.AND);
    searchCriteria.getCriteria().add(fieldCriterion);
    expenses = expenseRepository.readExpensesByCriteria(searchCriteria, 100,
    getPropertiesToRead());
}
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    //the list of all available properties displays below but to increase
    efficiency in your searches, you can omit unnecessary properties from your
    application and those values will not be returned
    props.add("shortDescription");
    props.add("uniqueKey");
    props.add("categories");
    props.add("createdOn");
    props.add("modifiedBy");
    props.add("createdBy");
    props.add("modifiedOn");
    props.add("version");
    props.add("contact");
    props.add("expenseDate");
    props.add("expensedBy");
    props.add("project");
    props.add("quantity");
    props.add("unitPrice");
```

```
    props.add("totalAmount");
    return props;
}
```

1.5.5.5.5 Posting expenses

The following sample posts an expense with given unique key.

Code Snippet

```
String expenseUniqueKey = "EXPE_1001";
protected ExpenseRepository expenseRepository;
expenseRepository.postExpense(expenseUniqueKey);
```

1.5.5.5.6 Voiding expenses

The following sample voids a expense with given unique key.

Code Snippet

```
String expenseUniqueKey = "EXPE_1001";
protected ExpenseRepository expenseRepository;
expenseRepository.voidExpense(expenseUniqueKey);
```

1.5.5.5.7 Deleting expenses

The following sample deletes an existing expense.

Code Snippet

```
//for this sample, it is assumed that an expense record already exists with a
unique key String equal to uniqueKey
protected ExpenseRepository expenseRepository;
public void test_deleteExpense() throws Exception {
    expenseRepository.deleteExpense(uniqueKey);
}
```

1.5.5.6 Group Accounts

The following .JAVA code samples show how you can use the Group Account Web Service to manage group account records. From the Group Account Web Services, you can create group accounts and set the group account name, description, and add users to the group. Group Account Web Services do not support all functionality available from the TeamConnect application, such as setting group record rights, setting group category rights, setting group tool rights, and setting default object views.

1.5.5.6.1 Creating group accounts

The following sample creates a group account record. Also see the section about User Accounts.

Code Snippet

```
// this sample assumes at least one user account record already exists with the
unique key (String) value equal to userUniqueKey
protected GroupAccountRepository groupAccountRepository;
private String insertGroupAccount() throws Exception {
    String groupAccountName = "administrators";
    GroupAccountCreate groupAccount = new GroupAccountCreate();
    //uniqueName is a required field
    groupAccount.setUniqueName(groupAccountName);
    //displayName is a required field
    groupAccount.setDisplayName(groupAccountName);
    //add users to the group
    groupAccount.getUserAccountUniqueKeys().add(userUniqueKeys);
    String uniqueKey = groupAccountRepository.insertGroupAccount(groupAccount);
    return uniqueKey;
}
private List<String> userAccountUniqueKeys() {
    List<String> userUniqueKeys = new List<String>();
    userUniqueKeys.add("USER_1005");
    userUniqueKeys.add("USER_1009");
    userUniqueKeys.add("USER_1041");
    return userUniqueKeys;
}
```

1.5.5.6.2 Updating group accounts

The following sample updates an existing group account record by changing the display name, changing the description, and removing an existing group member (user account).

Code Snippet

```
// this sample assumes a group account record already exists with the unique key
(String) value equal to uniqueKey
protected GroupAccountRepository groupAccountRepository;
public void test_updateGroupAccount() throws Exception {
    GroupAccountUpdate groupUpdate = new GroupAccountUpdate();
    //set the uniqueKey with the String value of the target group record to
    identify which record you are updating
    groupUpdate.setUniqueKey(uniqueKey);
    //update the existing display name
    groupUpdate.setDisplayName("BusinessAdministrators");
    //update the existing group description
```

```
    groupUpdate.setDescription("Administrators who manage Admin Settings and
    TeamConnect  user/group accounts");
    //remove a user from the group
    groupUpdate.getUserAccountDeleteUniqueKeys().add(userUniqueKeys);
    groupAccountRepository.updateGroupAccount(groupUpdate);
}
private List<String> userAccountUniqueKeys() {
    List<String> userUniqueKeys = new List<String>();
    userUniqueKeys.add("USER_1005");
    return userUniqueKeys;
}
```

1.5.5.6.3 Reading group accounts

The following sample reads the specified properties of a group account record. All available properties of a group account are listed but you can comment out or remove unnecessary properties in your application to improve performance.

Code Snippet

```
//define the list of group account record properties to return with the
readGroupAccount results
protected GroupAccountRepository groupAccountRepository;
private List<String> getProperties() {
    List<String> properties = new List<String>();
    properties.add("uniqueName");
    properties.add("displayName");
    properties.add("description");
    properties.add("createdBy");
    properties.add("createdOn");
    properties.add("modifiedBy");
    properties.add("modifiedOn");
    properties.add("users");
    return properties;
}
public GroupAccount test_readGroupAccount() throws Exception {
    //it is assumed that a group account record exists with the unique key equal to
    String, uniqueKey
    GroupAccount searchedGroup = groupAccountRepository.readGroupAccount(uniqueKey,
    getProperties());
    return searchedGroup;
}
```

1.5.5.6.4 Searching group accounts

The following sample searches groups by given search criteria and returns the specified properties of the resulting group account records.

Code Snippet

```
protected GroupAccountRepository groupAccountRepository;
private List<GroupAccount> test_readGroupAccountsByCriteria() throws
Exception {
    // Criterion for group account searching
    UserFieldCriterion fieldCriterion = new UserFieldCriterion();
    fieldCriterion.setComparator(UserComparator.ONE_OF);
    LegacySearchFieldPathExpression fieldPathExpression = new
    LegacySearchFieldPathExpression();
    fieldPathExpression.setSearchKeyPath("createdBy");
    fieldCriterion.setFieldPath(fieldPathExpression);
    fieldCriterion.getValues().add("JohnDoe");
    FieldSearchClause searchCriteria = new FieldSearchClause();
    searchCriteria.setOperator(LogicOperator.AND);
    searchCriteria.getCriteria().add(fieldCriterion);
    groups = groupAccountRepository.readGroupAccountsByCriteria(searchCriteria,
    100, getPropertiesToRead());
}
private List<String> getProperties() {
    List<String> properties = new List<String>();
    properties.add("uniqueName");
    properties.add("displayName");
    properties.add("description");
    properties.add("createdBy");
    properties.add("createdOn");
    properties.add("modifiedBy");
    properties.add("modifiedOn");
    properties.add("users");
    return properties;
}
```

1.5.5.6.5 Deleting group accounts

The following sample deletes a group account record with a given unique key.

Note: Deleting a group removes users from the group; however the user accounts are not deleted from the system. Also see the Administrator help, Deleting a Group, topic for Scenarios that Prevent Deletion.

Code Snippet

```
//for this sample it is assumed that a group record exists with a unique key String
equal to uniqueKey
protected GroupAccountRepository groupAccountRepository;
public void test_deleteGroupAccount() throws Exception {
    groupAccountRepository.deleteGroupAccount(uniqueKey);
}
```

1.5.5.7 History Records

The following .JAVA code samples show how you can use TeamConnect® Enterprise Web Services for functions related to history records.

1.5.5.7.1 Creating history records

The following sample creates a history record.

Code Snippet

```
Date date = new Date();
protected HistoryRepository historyRepository;
String uniqueKey = historyRepository.insertHistory(history);
private void createHistory() {
    HistoryCreate history = new HistoryCreate();
    //archivedOn is a required field
    history.setArchivedOn(date);
    //shortDescription is a required field
    history.setShortDescription("Subject - Update contact phone number");
    //text is an additional field that stores more information than the
    shortDescription; text allows up to 20000 characters
    history.setText("Additional information about the contact record update");
    Category cat = new Category();
    //the next two lines identify a category and then add the category to the
    history record
    cat.setUniqueKey("HIST_STAT");
    history.getCategories().add(cat);
    //use the parentObject property to associate the history with an existing
    record by unique key, for example, to describe a contact record that you have
    recently modified
    history.setParentObject("CONT_1234");
    return history;
}
```

1.5.5.7.2 Updating history records

The following sample updates a history record. Prerequisite to working with history records, you must also get the unique key value for the history record to update. If the history record is associated with another record, you need to get that record's unique key

Code Snippet

```
Date date = new Date();
protected HistoryRepository historyRepository;
historyRepository.updateHistory(history);
private void updateHistory() {
    HistoryUpdate history = new HistoryUpdate();
    //use setUniqueKey to identify the target history record to update
    history.setUniqueKey("HIST_0011");
    history.setShortDescription("Subject - Update contact phone number");
    history.setText("Additional information about the contact record update");
    history.setArchivedOn(date);
    Category cat = new Category();
    //the next two lines identify a category and then add the category to the
    history record
    cat.setUniqueKey("HIST_MANA");
    history.getCategories().add(cat);
    //use the parentObject property to associate the history with an existing
    record by unique key, for example, to describe a contact record that you have
    recently modified
    history.setParentObject("CONT_1234");
    return history;
}
```

1.5.5.7.3 Reading history records

The following sample reads a history record. You must get the unique key value for the history record to update.

Code Snippet

```
//for this sample, it is assumed that the unique key of the target
History record to read is a String, "HIST_0011"
protected HistoryRepository historyRepository;
History readHistory = historyRepository.readHistory("HIST_0011",
getPropertiesToRead());
private List<String> getPropertiesToRead() {
    List<String> properties = new List<String>();
    properties.add("version");
    properties.add("createdBy");
    properties.add("createdOn");
    properties.add("modifiedBy");
    properties.add("modifiedOn");
    properties.add("archivedOn");
    properties.add("text");
    properties.add("shortDescription");
    properties.add("parentObject");
```

```
    properties.add("categories");
    return properties;
}
```

1.5.5.7.4 Searching history records

The following sample searches for history records by short description (records that contain "updated contact record" in the Description) and date (records created after 12/01/2008).

Code Snippet

```
protected HistoryRepository historyRepository;
StringFieldCriterion fieldCriterion1 = new StringFieldCriterion();
fieldCriterion1.setComparator(StringComparator.CONTAINS);
LegacySearchFieldPathExpression fieldPathExpression1 = new
LegacySearchFieldPathExpression();
fieldPathExpression1.setSearchKeyPath("shortDescription");
fieldCriterion1.setFieldPath(fieldPathExpression);
fieldCriterion1.getValue().add("updated contact record");
DateFieldCriterion fieldCriterion2 = new DateFieldCriterion();
fieldCriterion2.setComparator(StringComparator.AFTER);
LegacySearchFieldPathExpression fieldPathExpression2 = new
LegacySearchFieldPathExpression();
fieldPathExpression2.setSearchKeyPath("createdOn");
fieldCriterion2.setFieldPath(fieldPathExpression);
fieldCriterion2.getValue().add(01-DEC-08);
FieldSearchClause searchCriteria = new FieldSearchClause();
searchCriteria.setOperator(LogicOperator.AND);
searchCriteria.getCriteria().add(fieldCriterion1);
searchCriteria.getCriteria().add(fieldCriterion2);
List<History> searchHistory =
historyRepository.readHistoriesByCriteria(searchCriteria, 100,
getPropertiesToRead());
private List<String> getPropertiesToRead() {
    List<String> properties = new List<String>();
    properties.add("version");
    properties.add("createdBy");
    properties.add("createdOn");
    properties.add("modifiedBy");
    properties.add("modifiedOn");
    properties.add("archivedOn");
    properties.add("text");
    properties.add("shortDescription");
    properties.add("parentObject");
    properties.add("categories");
    return properties;
}
```

1.5.5.7.5 Deleting history records

The following sample deletes a history record. You must get the unique key value for record to delete.

Code Snippet

```
//for this sample it is assumed that a history record exists with the unique key equal to a String, uniqueKey
protected HistoryRepository historyRepository;
historyRepository.deleteHistory(uniqueKey);
```

1.5.5.8 Invoices and Line Items

The following .JAVA code samples show how you can use TeamConnect® Enterprise Web Services for functions specific to invoice and line item records, such as adding line items to an invoice, posting, adjusting an invoice summary, and adjusting invoice line items.

1.5.5.8.1 Creating an Invoice with Line Items

The following example shows how to create an invoice and add an expense and fee line item to the invoice. When creating an invoice, you must populate a few required properties using the following methods (from the type class, InvoiceCreate): setNumberString, setInvoiceDate, and setVendorUniqueKey.

It is most practical to add one or more line items to an invoice. There are two types of line items you can add: expense or fee line items. For expense line items, you must populate a few required properties using the following methods (from the type class, LineItemCreate): setItemNumber, setServiceDate, setProjectUniqueKey, setOriginalRate, setOriginalQuantity, setOriginalDiscount, setType, and getCategories. For fee line items, you must populate a few required properties using the following methods (from the type class, LineItemCreate): setItemNumber, setServiceDate, setProjectUniqueKey, setOriginalRate, setOriginalQuantity, setOriginalDiscount, setType, getCategories, setTimekeeperUniqueKey, and setActivityUniqueKey.

Before starting, get the following:

- For expense line items, get the corresponding Line Item Expense Category Code(s)
- For fee line items, get the corresponding Line Item Fee Category Code(s) and Task Activity Lookup Table Activity Code(s)

When adding an expense line item to an invoice, use the Category object to specify the corresponding expense category. As described in the category example for contacts, you must already know the target expense category's unique key.

Code Snippet

```
protected InvoiceRepository invoice = new InvoiceRepository();
InvoiceCreate invoice = createInvoice();
invoice.insertInvoice(invoice);
private InvoiceCreate createInvoice() throws Exception {
    Date date = new Date();
    long time = date.getTime();
```

```
        InvoiceCreate invoice = new InvoiceCreate();
        //numberString is a required field
        invoice.setNumberString("invoice Web Service test " + time);
        //vendorUniqueKey is a required field
        invoice.setVendorUniqueKey("CONT_8094");
        //invoiceDate is a required field
        invoice.setInvoiceDate(date);
        invoice.setComment("comment");
        invoice.setPeriodStartDate(date);
        invoice.setPeriodEndDate(new Date(time + 1000000));
        invoice.setSubmittedElectronically(true);
        invoice.setReceivedDate(date);
        invoice.setTaxRate(new BigDecimal("8"));
        invoice.setWarnings("invoice warning");
        //Note that it is practical although not required to add one or more line items
        to an invoice
        invoice.getLineItemCreates().addAll(createLineItems());
        return invoice;
    }
    private List<LineItemCreate> createLineItems() throws Exception {
        List<LineItemCreate> lineItems = new List<LineItemCreate>();
        //create expense line item
        LineItemCreate lineItem1 = new LineItemCreate();
        //itemNumber is a required field
        lineItem1.setItemNumber(1);
        //for expense line items, originalQuantity is a required field that corresponds
        to a quantity of expense items being charged
        lineItem1.setOriginalQuantity(new BigDecimal("2"));
        //for expense line items, originalRate is a required field that corresponds to
        the per unit cost of expense item
        lineItem1.setOriginalRate(new BigDecimal("50"));
        //for expense line items, type (LineItemType) is a required field and would
        always be EXPENSE
        lineItem1.setType(LineItemType.EXPENSE);
        lineItem1.setServiceDate(date);
        //category for each line item is a required field
        Category cat1 = new Category();
        cat1.setUniqueKey("EXPE");
        lineItem1.getCategories().add(cat1);
        lineItems.add(lineItem1);
        //create fee line item
        LineItemCreate lineItem2 = new LineItemCreate();
        lineItem2.setItemNumber(2);
        //for Fee line items, the originalQuantity is a required field and will
        correspond to the hours worked by the timekeeper
        lineItem2.setOriginalQuantity(new BigDecimal("2"));
```

```
    //for Fee line items, the originalRate is a required field and will correspond
    to the associated timekeeper's rate
    lineItem2.setOriginalRate(new BigDecimal("50"));
    //for fee line items, type (LineItemType) is a required field and would always
    be FEE
    lineItem2.setType(LineItemType.FEE);
    lineItem2.setServiceDate(date);
    Category cat2 = new Category();
    cat2.setUniqueKey("TASK");
    lineItem2.getCategories().add(cat2);
    //for fee line items, the activityUniqueKey is a required field; the
    ActivityUniqueKey property needs to be constructed by "ACTI_"+<4 character Task
    Activity Lookup Table Activity Code>; see the example below where the Activity
    Code is A101
    lineItem2.setActivityUniqueKey("ACTI_A101");
    lineItems.add(lineItem2);
    return lineItems;
}
```

1.5.5.8.2 Updating and Adjusting an Invoice

The following example shows how to update an invoice. You can edit the properties of the invoice as well as adjust the invoice summary or adjust the invoice's line items when you perform an invoice update.

The section of the example that shows how to adjust an invoice summary does so by changing the invoice total to a specified amount. It is assumed that there is an existing invoice that will be adjusted. To access an existing invoice, use the InvoiceRepository's updateInvoice method. To create the invoice summary adjustment, use the type class, InvoiceAdjustmentCreate. Note that there are other ways to adjust an invoice summary and you can refer to the User Guide for more general information.

When creating an invoice summary adjustment, you must use the following set methods to populate required fields: setAdjustingUserUniqueKey(String value), setAdjustmentDate(Date value), setAdjustmentTarget(InvoiceAdjustmentTarget value), setAdjustmentValue(BigDecimal value), setAdjustmentMethod(AdjustmentMethod value).

For more information on adjusting line items, see Adjusting Invoice Line Items.

Code Snippet

```
//for this example, it is assumed that you already know the target
invoice record's unique key (where uniqueKey is a String variable)
protected InvoiceRepository invoice = new InvoiceRepository();
InvoiceUpdate invoice = updateInvoice();
invoice.updateInvoice(invoice);
private InvoiceUpdate updateInvoice() throws Exception {
    Date date = new Date();
    String uniqueKey = "inv5010897";
    InvoiceUpdate invoice = new InvoiceUpdate();
```

```
    //next line used to ID the invoice to update
    invoice.setUniqueKey(uniqueKey);
    //these 3 lines update invoice properties
    invoice.setNumberString("inv100-2053");
    invoice.setInvoiceDate(date);
    invoice.setVendorUniqueKey("ven2345");
    //these lines are used to adjust the invoice header/summary
    invoice.getAdjustmentUpdates().add(createInvoiceAdjustment());
    return invoice;
}
private InvoiceAdjustmentCreate createInvoiceAdjustment() throws Exception {
    String userUniqueKey = "user101";
    InvoiceAdjustmentCreate invoiceAdjustment = new InvoiceAdjustmentCreate();
    invoiceAdjustment.setAdjustingUserUniqueKey(userUniqueKey);
    invoiceAdjustment.setAdjustmentDate(date);
    invoiceAdjustment.setAdjustmentTarget(InvoiceAdjustmentTarget.TOTAL_INVOICE);
    invoiceAdjustment.setAdjustmentValue(2005.17);
    invoiceAdjustment.setAdjustmentMethod(AdjustmentMethod.NEW_AMOUNT);
    invoiceAdjustment.setInHouseComments("Rejected invoice because the timekeeper
    rate exceeds our agreement. Adjusting the total per acceptable rates.");
    invoiceAdjustment.setCommentsToVendor("Rejected invoice because the timekeeper
    rate exceeds our agreement. Adjusting the total per acceptable rates.");
    return invoiceAdjustment;
}
```

1.5.5.8.3 Adjusting Invoice Line Items

The following example shows how to adjust an invoice line item by changing the line item rate by a specified amount. In the example, it is assumed there is an existing invoice with one or more line items. To access an existing invoice, use the InvoiceRepository's updateInvoice method. To perform the line item adjustment, use the type class, LineItemAdjustmentUpdate.

Note: You can also adjust line items at the time you add line items to an invoice. In this case, you can use either the InvoiceRepository's insertInvoice or updateInvoice method. Since you are adjusting the line item and creating it at the same time, use the type class, LineItemAdjustmentCreate.

When performing a line item adjustment, you must use the following set methods to populate required fields: .

Code Snippet for adjusting invoice line items on invoice update

```
protected InvoiceRepository invoice = new InvoiceRepository();
InvoiceUpdate invoice = updateInvoice();
invoice.updateInvoice(invoice);
private InvoiceUpdate updateInvoice() throws Exception {
```

```
    Date date = new Date();
    InvoiceUpdate invoice = new InvoiceUpdate();
    //these 3 lines are used to ID the invoice to update
    invoice.setNumberString("inv100-2053");
    invoice.setInvoiceDate(date);
    invoice.setVendorUniqueKey("ven2345");
    //these lines are used to adjust an invoice line item
    invoice.getLineItemCreates().add(createLineItemAdjustment());
    return invoice;
}
private LineItemAdjustmentCreate createLineItemAdjustment() throws Exception {
    String userUniqueKey = "user_101";
    LineItemAdjustmentCreate lineItemAdjustment = new LineItemAdjustmentCreate();
    lineItemAdjustment.setAdjustingUserUniqueKey(userUniqueKey);
    lineItemAdjustment.setAdjustmentDate(date);
    lineItemAdjustment.setAdjustmentTarget(LineItemAdjustmentTarget.RATE);
    lineItemAdjustment.setAdjustmentValue(45.5);
    lineItemAdjustment.setAdjustmentMethod(AdjustmentMethod.REDUCE_BY_AMOUNT);
    lineItemAdjustment.setInHouseComments("Rejected invoice because the timekeeper
    rate exceeds our agreement. Adjusting the rate down.");
    lineItemAdjustment.setCommentsToVendor("Rejected invoice because the timekeeper
    rate exceeds our agreement. Adjusting the line item rate.");
    return lineItemAdjustment;
}
```

1.5.5.8.4 Reading an Invoice

The following sample reads an invoice record. You must get the unique key value for the invoice record to update.

Code Snippet

```
//for this sample, it is assumed that the unique key of the target Invoice record
to read is a String, "INVC_0011"
protected InvoiceRepository invoiceRepository;
Invoice invoice = invoiceRepository.readInvoice("INVC_0011",
getPropertiesToRead());
private List<String> getPropertiesToRead() {
    List<String> properties = new List<String>();
    properties.add("version");
    properties.add("createdBy");
    properties.add("createdOn");
    properties.add("modifiedBy");
    properties.add("modifiedOn");
    properties.add("numberString");
    properties.add("invoiceDate");
    properties.add("receivedDate");
```

```
    properties.add("periodStartDate");
    properties.add("periodEndDate");
    properties.add("submittedElectronically");
    properties.add("postingStatus");
    properties.add("vendor");
    properties.add("submittedTotal");
    properties.add("currency");
    properties.add("comment");
    properties.add("warnings");
    properties.add("taxRate");
    properties.add("adjustments");
    properties.add("lineItems");
    properties.add("originalExpenseTotal");
    properties.add("originalFeeTotal");
    properties.add("originalInvoiceTotal");
    properties.add("expenseDiscountTotal");
    properties.add("feeDiscountTotal");
    properties.add("invoiceDiscountTotal");
    properties.add("expenseAdjustmentTotal");
    properties.add("feeAdjustmentTotal");
    properties.add("invoiceAdjustmentTotal");
    properties.add("expenseTaxTotal");
    properties.add("feeTaxTotal");
    properties.add("invoiceTaxTotal");
    properties.add("netExpenseTotal");
    properties.add("netFeeTotal");
    properties.add("netInvoiceTotal");
    properties.add("categories");
    properties.add("note");
    return properties;
}
```

1.5.5.8.5 Searching for Invoices

The following sample searches invoices by given search criteria (a particular invoice vendor) and returns the specified properties of the resulting invoice records.

Code Snippet

```
protected InvoiceRepository invoiceRepository;
private List<Invoice> test_readInvoicesByCriteria() throws Exception
{
    // Criterion for invoice searching
    StringFieldCriterion fieldCriterion = new StringFieldCriterion();
    fieldCriterion.setComparator(UserComparator.CONTAINS);
```

```
        LegacySearchFieldPathExpression fieldPathExpression = new
        LegacySearchFieldPathExpression();
        fieldPathExpression.setSearchKeyPath("vendor");
        fieldCriterion.setFieldPath(fieldPathExpression);
        fieldCriterion.getValue().add("Rubidoux Accounting");
        FieldSearchClause searchCriteria = new FieldSearchClause();
        searchCriteria.setOperator(LogicOperator.AND);
        searchCriteria.getCriteria().add(fieldCriterion);
        invoices = invoiceRepository.readInvoicesByCriteria(searchCriteria, 100,
        getPropertiesToRead());
    }
    private List<String> getPropertiesToRead() {
        List<String> properties = new List<String>();
        properties.add("version");
        properties.add("createdBy");
        properties.add("createdOn");
        properties.add("modifiedBy");
        properties.add("modifiedOn");
        properties.add("numberString");
        properties.add("invoiceDate");
        properties.add("receivedDate");
        properties.add("periodStartDate");
        properties.add("periodEndDate");
        properties.add("submittedElectronically");
        properties.add("postingStatus");
        properties.add("vendor");
        properties.add("submittedTotal");
        properties.add("currency");
        properties.add("comment");
        properties.add("warnings");
        properties.add("taxRate");
        properties.add("adjustments");
        properties.add("lineItems");
        properties.add("originalExpenseTotal");
        properties.add("originalFeeTotal");
        properties.add("originalInvoiceTotal");
        properties.add("expenseDiscountTotal");
        properties.add("feeDiscountTotal");
        properties.add("invoiceDiscountTotal");
        properties.add("expenseAdjustmentTotal");
        properties.add("feeAdjustmentTotal");
        properties.add("invoiceAdjustmentTotal");
        properties.add("expenseTaxTotal");
        properties.add("feeTaxTotal");
        properties.add("invoiceTaxTotal");
```

```
        properties.add("netExpenseTotal");
        properties.add("netFeeTotal");
        properties.add("netInvoiceTotal");
        properties.add("categories");
        properties.add("note");
        return properties;
    }
```

1.5.5.8.6 Posting an Invoice

The following example shows how to post an invoice. Before starting, you must get the unique key of the target invoice record.

Code Snippet

```
String invoiceUniqueKey = "inv_1001";
protected InvoiceRepository invoiceRepository;
invoiceRepository.postInvoice(invoiceUniqueKey);
```

1.5.5.8.7 Voiding an Invoice

The following example shows how to void an invoice. Before starting, you must get the unique key of the target invoice record.

Code Snippet

```
String invoiceUniqueKey = "inv_1001";
protected InvoiceRepository invoiceRepository;
invoiceRepository.voidInvoice(invoiceUniqueKey);
```

1.5.5.8.8 Searching for Line Items

When searching for or filtering line items that are associated with an invoice, use the Invoice Repository's readInvoicesByCriteria method. In the definition of the fieldPathExpression.setSearchKeyPath, use the properties defined in the type class, LineItem, to identify which line item fields to use as search criteria.

The following example filters an invoice's line items based on associated project, fee line items, and a given timekeeper.

Code Snippet

```
//for this example, it is assumed that at least one invoice line item record that
meets the search criterion already exists
protected InvoiceRepository invoiceRepository;
public void searchInvoices() throws Exception {
    ObjectFieldCriterion fieldCriterion1 = new ObjectFieldCriterion();
    fieldCriterion.setComparator(ObjectComparator.EQUALS);
    LegacySearchFieldPathExpression fieldPathExpression1 = new
    LegacySearchFieldPathExpression();
```

```
        fieldPathExpression1.setSearchKeyPath("lineItemList.project.primaryKey");
        fieldCriterion1.setFieldPath(fieldPathExpression1);
        fieldCriterion.setValue("DISP_1007");
        FieldSearchClause searchCriteria = new FieldSearchClause();
        searchCriteria.setOperator(LogicOperator.AND);
        searchCriteria.getCriteria().add(fieldCriterion1);
        //the following section defines a second fieldCriterion item for filtering fee
        line items
        StringFieldCriterion fieldCriterion2 = new StringFieldCriterion();
        fieldCriterion.setComparator(StringComparator.EQUALS_ENFORCE_CASE);
        LegacySearchFieldPathExpression fieldPathExpression2 = new
        LegacySearchFieldPathExpression();
        fieldPathExpression2.setSearchKeyPath("lineItem.type");
        fieldCriterion2.setFieldPath(fieldPathExpression2);
        fieldCriterion.setValue("FEE");
        searchCriteria.getCriteria().add(fieldCriterion2);
        //the following section defines a third fieldCriterion item for filtering line
        items associated with a given timekeeper
        StringFieldCriterion fieldCriterion3 = new StringFieldCriterion();
        fieldCriterion.setComparator(StringComparator.EQUALS_ENFORCE_CASE);
        LegacySearchFieldPathExpression fieldPathExpression3 = new
        LegacySearchFieldPathExpression();
        fieldPathExpression3.setSearchKeyPath("lineItem.timekeeper.uniqueKey");
        fieldCriterion3.setFieldPath(fieldPathExpression3);
        fieldCriterion.setValue("CONT_1010");
        searchCriteria.getCriteria().add(fieldCriterion3);
        List<Invoice> invoices =
        invoiceRepository.readInvoicesByCriteria(searchCriteria, 100,
        getPropertiesToRead());
    }
    private List<String> getPropertiesToRead() {
        List<String> properties = new List<String>();
        properties.add("numberString");
        properties.add("invoiceDate");
        properties.add("vendor");
        properties.add("lineItems.version");
        properties.add("lineItems.itemNumber");
        properties.add("lineItems.serviceDate");
        properties.add("lineItems.version");
        properties.add("lineItems.originalRate");
        properties.add("lineItems.originalQuantity");
        properties.add("lineItems.originalDiscount");
        properties.add("lineItems.originalTotal");
        properties.add("lineItems.adjustedTotal");
        return properties;
    }
```

1.5.5.8.9 Deleting an Invoice

The following sample deletes an invoice record. You must get the unique key value for record to delete.

Code Snippet

```
//for this sample it is assumed that an invoice record exists with the unique key equal to a String, uniqueKey
protected InvoiceRepository invoiceRepository;
invoiceRepository.deleteInvoice(uniqueKey);
```

1.5.5.9 Projects (Matters)

The following .JAVA code samples show how you can use TeamConnect® Enterprise Web Services for functions specific to project or custom object records, such as changing a project record's phase. Prerequisite to working with project records, the Custom Object Definition must already exist in TeamConnect . In addition, you need to get the following information before writing an application using Web Services:

- The project's Unique Code
- The unique keys of the project's phases
- The defined phase transitions for the project

Also see Working with Categories and Working with Custom Fields for additional information to gather before writing an application.

1.5.5.9.1 Creating a Project With Assignees and a Custom Field

Creating a project record is done using the ProjectRepository object. For example, use the ProjectRepository's insertProject (ProjectCreate project) method. The resulting record's unique key is returned. The insertProject call will result in a SOAP/HTTP request and requires authentication.

When creating a project you can also add one or more assignees. From the type class, ProjectCreate, you need to use the getAssigneeCreates() method (see the example below for how to add items to the resulting list of ProjectAssigneeCreate).

Before you can start, you must get the unique key for the target assignee role and the unique key for the user to add as assignee either from the TeamConnect Designer GUI or from the database.

Note: *After you understand how to add assignees to a project, you can apply a similar concept to the procedure for adding relations to a project.*

Code Snippet

```
//in the sample below, basic require fields are commented but it is recommended to verify whether additional required fields have been defined from the TeamConnect custom object definition or from the responsible solution developer
protected ProjectRepository projectRepository;
ProjectCreate project = createProject();
projectRepository.insertProject(project);
```

```
private ProjectCreate createProject() throws Exception {
    Date date = new Date();
    //the EntityTypeUniqueKey is an object's 4-character unique code; look this
    value up either in the Designer area Object Definitions list or from the
    database
    String Entity_Type_Unique_Key = "DISP";
    ProjectCreate project = new ProjectCreate();
    //entityTypeUniqueKey is a required field
    project.setEntityTypeUniqueKey(Entity_Type_Unique_Key);
    //name is a required field
    project.setName("Small Claims Forster vs. Gloucster");
    //idNumber is a required field
    project.setIdNumber("Dispute_1001");
    project.setClosedOn(date);
    //the next three lines add a category to the project
    Category cat = new Category();
    cat.setUniqueKey("DISP EXTE");
    project.getCategories().add(cat);
    //the next four lines create a placeholder for an existing Text custom field,
    identify the known custom field name (statusSummary), set the custom field
    value, and add the custom field placeholder to its parent category
    TextCustomField statusSummary = new TextCustomField();
    statusSummary.setFieldName("statusSummary");
    statusSummary.setValue("Dispute has been resolved. Settlement reached.");
    cat.getTextCustomFields().add(statusSummary);
    project.getAssigneeCreates().addAll(getAssigneeCreates());
    return project;
}
public List<ProjectAssigneeCreate> getAssigneeCreates() {
    Date date = new Date();
    List<ProjectAssigneeCreate> assignees = new List<ProjectAssigneeCreate>();
    ProjectAssigneeCreate assignee1 = new ProjectAssigneeCreate();
    assignee1.setRoleUniqueKey("RPRT");
    assignee1.setAssignedOn(date);
    assignee1.setUnassignedOn(date);
    assignee1.setUserUniqueKey("USER9614");
    assignee1.setActive(True);
    assignees.add (assignee1);
    ProjectAssigneeCreate assignee2 = new ProjectAssigneeCreate();
    assignee2.setRoleUniqueKey("RPRT");
    assignee2.setAssignedOn(date);
    assignee2.setUnassignedOn(date);
    assignee2.setUserUniqueKey("USER9630");
    assignee2.setActive(False);
    assignees.add (assignee2);
    return assignees;
```

```
}
```

1.5.5.9.2 Updating a Project by Adding an Embedded Object Record

Updating a project is similar to updating other record types, such as contacts. You need to use the ProjectRepository's updateProject method. In addition, from the type class, ProjectUpdate, use provided methods to identify the target record and update its property values. Before you can start, you need to know the target project record's unique key.

You can add one or more embedded object records either at the time you update or create a project record. The example below illustrates how to add an embedded object record (Damage) to an existing project record (Dispute).

Note: *Adding an embedded object record to a project is analogous to adding an assignee or relation to a project.*

Code Snippet

```
protected ProjectRepository projectRepository;
ProjectUpdate project = updateProject();
projectRepository.updateProject(project);
private ProjectUpdate updateProject() throws Exception {
    String date = String.valueOf(new Date().getTime());
    //the EntityTypeUniqueKey is an object's 4-character unique code; look this
    value up either in the Designer area Object Definitions list or from the
    database
    String Entity_Type_Unique_Key = "DISP";
    ProjectUpdate project = new ProjectUpdate();
    //the next line identifies which record to update by its unique key
    project.setUniqueKey("DISP_2468");
    project.setEntityTypeUniqueKey(Entity_Type_Unique_Key);
    project.setName("Small Claims Forster vs. Gloucster");
    project.setClosedOn(date);
    project.setIdNumber("Dispute_1001");
    //the next three lines add a category to the project
    Category cat = new Category();
    //the category must already be created in TeamConnect ; you need to get the
    unique key from the TeamConnect  GUI or database
    cat.setUniqueKey("DISP EXTE");
    project.getCategories().add(cat);
    project.getEmbeddedEntityCreates().add(getEmbeddedEntityCreates());
    return project;
}
public EmbeddedEntityCreate getEmbeddedEntityCreates() {
    Date date = new Date();
    // List<EmbeddedEntityCreate> embeddedEntities = new
    List<EmbeddedEntityCreate>();
    EmbeddedEntityCreate embeddedEntity = new EmbeddedEntityCreate();
```

```
    embeddedEntity.setName("Damage_1002");
    //the embedded object or project object definition must already exist in
    TeamConnect ; you need to get the embedded object's 4-character unique key
    (Unique Code) from the TeamConnect  GUI or database
    embeddedEntity.setEntityTypeUniqueKey("DAMA");
    embeddedEntity.setContactUniqueKey("cont_1004");
    embeddedEntity.setIdNumber("Damage_1002");
    Category catEmbedded = new Category();
    //the embedded object's category must already exist in TeamConnect; you need to
    get the corresponding category's unique key from the TeamConnect GUI or
    database
    catEmbedded.setUniqueKey("DAMG");
    embeddedEntity.getCategories().add(catEmbedded);
    return embeddedEntity;
}
```

1.5.5.9.3 Changing a Project's Phase

When changing a project's phase, you must have already gathered the record's unique key as well as the unique key of the target phase to transition to. The ProjectRepository changePhase method will be validated against existing Phase Transition definitions for the project object definition.

Code Snippet

```
proj.Repository.changePhase("DISP_1004", "OPNN_OPEN");
```

1.5.5.9.4 Reading Projects

Reading a project record is similar to reading a contact record. A main difference would be when you request the categories property in the resulting project record's returned property values. By default, the returned list of categories also stores all property values associated with all project categories.

Code Snippet

```
//for this example, it is assumed that you already know the target project record's
unique key (where uniqueKey is a String variable)
protected ProjectRepository projectRepository;
public void readProject() throws Exception {
    Project project = (Project)
    projectRepository.readProject(uniqueKey, getPropertiesToRead());
}
private List<String> getPropertiesToRead() {
    List<String> properties = new List<String>();
    properties.add("name");
    properties.add("idNumber");
    properties.add("currentPhase.storedValue");
    properties.add("assignees");
    properties.add("categories");
```

```
    return properties;
}
```

1.5.5.9.5 Reading Child Projects for a Record

The sample below illustrates how you can read child projects of a project record by the child project type (project's 4-character unique code).

Code Snippet

```
// in this sample, it is assumed there is an existing parent project record with a unique key String, parentKey
// it is also assumed that there are one or more child project records (where the custom object definition's unique code is "CLAI")
protected ProjectRepository projectRepository;
public void readChildProjectsForEntityType() throws Exception {
    // in the list of child Projects returned, only the name properties will be returned
    List<Project> children =
    projectRepository.readChildProjectsForEntityType(parentKey, "CLAI",
    Arrays.asList("name"));
}
```

1.5.5.9.6 Searching for Projects

Searching for projects is similar to searching for other object types. The following example illustrates how to search among projects for the following criteria: a given project phase, and given main assignee (by their user name).

Code Snippet

```
protected ProjectRepository projectRepository;
public void searchProjects() throws Exception {
    StringFieldCriterion fieldCriterion1 = new StringFieldCriterion();
    fieldCriterion.setComparator(StringComparator.EQUALS_ENFORCE_CASE);
    LegacySearchFieldPathExpression fieldPathExpression1 = new
    LegacySearchFieldPathExpression();
    fieldPathExpression1.setSearchKeyPath("currentPhase.storedValue");
    fieldCriterion1.setFieldPath(fieldPathExpression1);
    //need clarification if unique key or name
    //if unique key, then filtering for projects that are in phase, Closed
    fieldCriterion.getValues().add("CLOS");
    FieldSearchClause searchCriteria = new FieldSearchClause();
    searchCriteria.setOperator(LogicOperator.AND);
    searchCriteria.getCriteria().add(fieldCriterion1);
    //the following section defines a second fieldCriterion item for
    StringFieldCriterion fieldCriterion2 = new StringFieldCriterion();
    fieldCriterion.setComparator(StringComparator.EQUALS);
```

```
    LegacySearchFieldPathExpression fieldPathExpression2 = new
    LegacySearchFieldPathExpression();
    fieldPathExpression2.setSearchKeyPath("mainAssignee.user.username");
    fieldCriterion2.setFieldPath(fieldPathExpression2);
    fieldCriterion.getValue().add("JohnDoe");
    searchCriteria.getCriteria().add(fieldCriterion2);
    List<Project> projects =
    projectRepository.readProjectsByCriteria(searchCriteria, 100,
    getPropertiesToRead());
}
private List<String> getPropertiesToRead() {
    List<String> properties = new List<String>();
    properties.add("name");
    properties.add("idNumber");
    return properties;
}
```

1.5.5.9.7 Deleting a Project

The following sample deletes a project record. You must get the unique key value for record to delete.

Code Snippet

```
//for this sample it is assumed that a project record exists with the unique key
equal to a String, uniqueKey
protected ProjectRepository projectRepository;
projectRepository.deleteProject(uniqueKey);
```

1.5.5.10 Involved Records

An involved is a unique record typically associated with a matter or project.

1.5.5.10.1 Inserting Involved records

Inserts or creates an involved party record (associated with a project/matter). Often, an involved will be created and added to a project record at the same time as the project creation. This example assumes a project has already been created.

Code Snippet

```
protected InvolvedRepository involvedRepository;
String uniqueKey = Repository.insertInvolved(involved);
private void createInvolved() {
    InvolvedCreate involved = new InvolvedCreate();
    //the following are required fields
    //in the following line, "DINV" is the unique code of the Involved object
    definition, which is a child of the related Dispute (project) object definition
    involved.setEntityTypeUniqueKey("DINV");
```

```
    //in the following line, "DISP_2008" is the existing dispute record unique key
    involved.setProjectUniqueKey("DISP_2008");
    //in the following line, "CONT_5497" corresponds to the unique key of the
    contact record for the person you are adding as the involved to a project
    involved.setContactUniqueKey("CONT_5497");
    involved.setActive(true);
    Category cat = new Category();
    //the next two lines identify a category and then add the category to the
    involved record; this category corresponds to the involved party's default role
    cat.setUniqueKey("DINV_MNGR");
    involved.getCategories().add(cat);
    return involved;
}
```

1.5.5.10.2 Updating Involved records

Updating an involved is similar to updating other record types, such as contacts, except that you will be updating only the involved record's categories, custom field values, or clearing existing property values. You need to use the InvolvedRepository's updateInvolved method. Before you can start, you need to know the target record's unique key.

Code Snippet

```
private InvolvedRepository involvedRepository;
Date date = new Date();
InvolvedUpdate involved = new InvolvedUpdate();
Category cat = new Category();
//the next two lines identify a category and then add the category to the involved
record
cat.setUniqueKey("INPA_ARBI");
involved.getCategories().add(cat);
Category cat2 = new Category();
involved.getCategories().add(cat2);
cat2.setUniqueKey("INPA_MEDI");
//the next four lines create a placeholder for an existing DateTime custom field,
identify the known custom field name (InactiveDateDI), set the custom field value,
and add the custom field placeholder to its parent category
DateTimeCustomField dateTime1 = new DateTimeCustomField();
dateTime1.setFieldName("InactiveDateDI");
dateTime1.setValue(date);
cat2.getDateTimeCustomFields().add(dateTime1);
DateTimeCustomField dateTime2 = new DateTimeCustomField();
dateTime2.setFieldName("InvolvementDateDI");
dateTime2.setValue(date);
cat2.getDateTimeCustomFields().add(dateTime2);
involvedRepository.updateInvolved(involved);
```

1.5.5.10.3 Reading Involved records

Reading an involved record is similar to reading a contact record.

Code Snippet

```
//for this sample, it is assumed there exists an involved record with the unique
key equal to a String, invUniqueKey
private InvolvedRepository involvedRepository;
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    props.add("uniqueKey");
    props.add("contact");
    props.add("project");
    props.add("project.name");
    props.add("contact.uniqueKey");
    return props;
}
public void test_readInvolved throws Exception {
    // Search the involved record
    Involved involved = involvedRepository.readInvolved(invUniqueKey,
    getPropertiesToRead());
}
```

1.5.5.10.4 Reading Involved Parties for a Project Record

The following sample gets requested properties for involved party records that are associated with a project record, given the unique key of the project record.

Code Snippet

```
readInvolvedsForProject
private InvolvedRepository involvedRepository;
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    props.add("uniqueKey");
    props.add("contact");
    props.add("project");
    props.add("project.name");
    props.add("contact.uniqueKey");
    return props;
}
public void test_readInvolvedsForProject throws Exception {
    // Search the involved records for a project
    //in the following operation, "DISP_5941" is the unique key of the project
    record whose involved party records will be returned
    List<Involved> involvedParties =
    involvedRepository.readInvolvedsForProject("DISP_5941",
```

```
    getPropertiesToRead());
}
```

1.5.5.10.5 Searching Involved Party Records

The following sample searches involved party records by entity type and contact last name.

Code Snippet

```
//for this sample, it is assumed there exists an involved record
with the unique key equal to a String, invUniqueKey
private InvolvedRepository involvedRepository;
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    props.add("uniqueKey");
    props.add("person.lastName");
    props.add("person.firstName");
    props.add("project");
    props.add("project.name");
    props.add("contact.uniqueKey");
    return props;
}
public void searchInvolveds() throws Exception {
    StringFieldCriterion fieldCriterion1 = new StringFieldCriterion();
    fieldCriterion.setComparator(StringComparator.EQUALS_ENFORCE_CASE);
    LegacySearchFieldPathExpression fieldPathExpression1 = new
    LegacySearchFieldPathExpression();
    fieldPathExpression1.setSearchKeyPath("entityType.uniqueCode");
    fieldCriterion1.setFieldPath(fieldPathExpression1);
    //filtering for involved party records defined by the Involved object
    definition with the TeamConnect  unique code, "DINV"
    fieldCriterion.getValue().add("DINV");
    FieldSearchClause searchCriteria = new FieldSearchClause();
    searchCriteria.setOperator(LogicOperator.AND);
    searchCriteria.getCriteria().add(fieldCriterion1);
    //the following section defines a second fieldCriterion item for
    StringFieldCriterion fieldCriterion2 = new StringFieldCriterion();
    fieldCriterion.setComparator(StringComparator.EQUALS);
    LegacySearchFieldPathExpression fieldPathExpression2 = new
    LegacySearchFieldPathExpression();
    fieldPathExpression2.setSearchKeyPath("contact.person.lastName");
    fieldCriterion2.setFieldPath(fieldPathExpression2);
    fieldCriterion.getValue().add("Doe");
    searchCriteria.getCriteria().add(fieldCriterion2);
    public void test_readInvolvedsByCriteria throws Exception {
    // Search the involved records
```

```
    Involved involved = involvedRepository.readInvolvedsByCriteria(searchCriteria,
    100,
    getPropertiesToRead());
}
```

1.5.5.10.6 Deleting an Involved

The following sample deletes an involved party record. You must get the unique key value for record to delete.

Code Snippet

```
//for this sample it is assumed that an involved party record exists with the
unique key equal to a String, uniqueKey
protected InvolvedRepository involvedRepository;
involvedRepository.deleteInvolved(uniqueKey);
```

1.5.5.11 Tasks

The following .JAVA code samples show how you can use TeamConnect® Enterprise Web Services for functions related to task records.

1.5.5.11.1 Creating tasks

The following sample creates a task record. Prerequisite to working with task records, you must also get the unique key value for an existing TeamConnect user record to add as a task assignee.

Code Snippet

```
protected TaskRepository taskRepository;
private String createTask() throws Exception {
    Date date = new Date();
    TaskAssigneeCreate taskAssignee = new TaskAssigneeCreate();
    taskAssignee.setUserUniqueKey("user_1003");
    taskAssignee.setAssignedOn(date);
    TaskCreate task = new TaskCreate();
    //the shortDescription field is required
    task.setShortDescription("subject");
    task.setActualHours(new BigDecimal("8.00"));
    task.setCurrentAssignee(taskAssignee);
    task.setPriority(TaskPriority.HIGH);
    task.setEstimatedHours(new BigDecimal("9.00"));
    task.setStartDate(date);
    task.setNote("A_task_note_" + date.getTime());
    task.setRateAmount(new BigDecimal("50"));
    return taskRepository.insertTask(task);
}
```

1.5.5.11.2 Updating tasks

The following sample updates a task record with the actual hours to complete the task, a newly calculated total amount, the date the task was completed, and percent completion value.

Code Snippet

```
//for this sample, it is assumed that a task record already exists with a unique key String equal to uniqueKey
protected TaskRepository taskRepository;
public void test_updateTask() throws Exception {
Date date = new Date();
TaskUpdate taskUpdate = new TaskUpdate();
//use the setUniqueKey method to identify the target task record to update by its unique key
taskUpdate.setUniqueKey(uniqueKey);
taskUpdate.setActualHours(new BigDecimal("15.00"));
taskUpdate.setRateAmount(new BigDecimal("50"));
taskUpdate.setCompletedDate(date);
taskUpdate.setCompletedPercent(100.00);
taskRepository.updateTask(taskUpdate);
}
```

1.5.5.11.3 Reading tasks

The following sample reads a task record and returns the specified property values.

Code Snippet

```
protected TaskRepository taskRepository;
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    //the list of all available properties displays below but to increase efficiency in your searches, you can omit unnecessary properties from your application and those values will not be returned
    props.add("shortDescription");
    props.add("uniqueKey");
    props.add("currentAssignee");
    props.add("categories");
    props.add("createdOn");
    props.add("modifiedBy");
    props.add("version");
    props.add("modifiedOn");
    props.add("contact");
    props.add("project");
    props.add("activityItem");
    props.add("priority");
    props.add("workStatus");
```

```
    props.add("postingStatus");
    props.add("dueDate");
    props.add("startDate");
    props.add("completedDate");
    props.add("completedPercent");
    props.add("actualHours");
    props.add("estimatedHours");
    props.add("rateAmount");
    props.add("totalAmount");
    props.add("note");
    return props;
}
private Task readTask() throws Exception {
    Task readTask = taskRepository.readTask(uniqueKey, getPropertiesToRead());
    return readTask;
}
```

1.5.5.11.4 Searching tasks

The following sample searches for one or more task records and returns the specified property values. For this sample, it is assumed there is one or more tasks already existing with a subject String field value equal to subject.

Code Snippet

```
protected TaskRepository taskRepository;
private List<Task> test_readTasksByCriteria() throws Exception {
    // Criterion for task searching
    StringFieldCriterion fieldCriterion = new StringFieldCriterion();
    fieldCriterion.setComparator(StringComparator.EQUALS_ENFORCE_CASE);
    LegacySearchFieldPathExpression fieldPathExpression = new
    LegacySearchFieldPathExpression();
    fieldPathExpression.setSearchKeyPath("shortDescription");
    fieldCriterion.setFieldPath(fieldPathExpression);
    fieldCriterion.getValue().add(subject);
    FieldSearchClause searchCriteria = new FieldSearchClause();
    searchCriteria.setOperator(LogicOperator.AND);
    searchCriteria.getCriteria().add(fieldCriterion);
    tasks = taskRepository.readTasksByCriteria(searchCriteria, 100,
    getPropertiesToRead());
}
private List<String> getPropertiesToRead() {
    List<String> props = new List<String>();
    //the list of all available properties displays below but to increase
    efficiency in your searches, you can omit unnecessary properties from your
    application and those values will not be returned
    props.add("shortDescription");
```

```
        props.add("uniqueKey");
        props.add("currentAssignee");
        props.add("categories");
        props.add("createdOn");
        props.add("modifiedBy");
        props.add("version");
        props.add("modifiedOn");
        props.add("contact");
        props.add("project");
        props.add("activityItem");
        props.add("priority");
        props.add("workStatus");
        props.add("postingStatus");
        props.add("dueDate");
        props.add("startDate");
        props.add("completedDate");
        props.add("completedPercent");
        props.add("actualHours");
        props.add("estimatedHours");
        props.add("rateAmount");
        props.add("totalAmount");
        props.add("note");
        return props;
    }
```

1.5.5.11.5 Posting tasks

The following sample posts a task with given unique key.

Code Snippet

```
String taskUniqueKey = "TASK_1001";
protected TaskRepository taskRepository;
taskRepository.postTask(taskUniqueKey);
```

1.5.5.11.6 Voiding tasks

The following sample voids a task with given unique key.

Code Snippet

```
String taskUniqueKey = "TASK_1001";
protected TaskRepository taskRepository;
taskRepository.voidTask(taskUniqueKey);
```

1.5.5.11.7 Reassigning tasks

The following sample reassigns a task from a current assignee (user) to a different assignee.

Code Snippet

```
//for this sample, it is assumed that a task record already exists with a unique
key String equal to uniqueKey
protected TaskRepository taskRepository;
private test_reassign() throws Exception {
    //reassign a task identified by its unique key to a new assignee, identified by
    his corresponding user unique key ("user_1005")
    taskRepository.reassign(uniqueKey, "user_1005");
}
```

1.5.5.11.8 Deleting tasks

The following sample deletes an existing task.

Code Snippet

```
//for this sample it is assumed that a task record exists with a unique key String
equal to uniqueKey
protected TaskRepository taskRepository;
public void test_deleteTask() throws Exception {
    taskRepository.deleteTask(uniqueKey);
}
```

1.5.5.12 User Accounts

The following .JAVA code samples show how you can use TeamConnect® Enterprise Web Services for functions related to user records.

1.5.5.12.1 Creating user accounts

The following sample creates a user account record. Prerequisite to working with user records, a corresponding contact (type person) record must already exist in TeamConnect . In addition, you need to get the following information before writing an application using Web Services: the unique key of the related contact record.

Code Snippet

```
// this sample assumes a contact record (type person) already exists with the
unique key (String) value equal to contactUniqueKey
protected UserAccountRepository userAccountRepository;
private String insertUserAccount() throws Exception {
    String userAccountName = "John_Doe";
    UserAccountCreate userAccount = new UserAccountCreate();
    //username is a required field
    userAccount.setUsername(userAccountName);
```

```
    //password is a required field
    userAccount.setPassword("p@ssw0rd");
    //contactUniqueKey is a required field
    userAccount.setContactUniqueKey(contactUniqueKey);
    userAccount.setShortDescription("new user hired 10/01/2008, business
    administrator");
    userAccount.setUserType(UserType.NORMAL);
    userAccount.setActive(true);
    String uniqueKey = userAccountRepository.insertUserAccount(userAccount);
    return uniqueKey;
}
```

1.5.5.12.2 Updating user accounts

The following sample updates an existing user account record by changing the password, changing the user type, and inactivating the user account.

Note: *In actual business usage, if you are inactivating a user account, it is unlikely you would need to change other properties but the samples are provided for you to use as necessary.*

Code Snippet

```
// this sample assumes a user record already exists with the unique key (String)
value equal to uniqueKey
protected UserAccountRepository userAccountRepository;
public void test_updateUserAccount() throws Exception {
    UserAccountUpdate userUpdate = new UserAccountUpdate();
    //set the uniqueKey with the String value of the target user record to identify
    which record you are updating
    userUpdate.setUniqueKey(uniqueKey);
    //update the existing password
    userUpdate.setPassword("newP@ssw0rd");
    //update the existing userType
    userUpdate.setUserType(UserType.LIMITED);
    //inactive the user account
    userUpdate.setActive(false);
    userAccountRepository.updateUserAccount(userUpdate);
}
```

1.5.5.12.3 Reading user accounts

The following sample reads the specified properties of a user account record.

Code Snippet

```
//define the list of user record properties to return with the readUserAccount
results
protected UserAccountRepository userAccountRepository;
```

```
private List<String> getProperties() {
    List<String> properties = new List<String>();
    properties.add("username");
    properties.add("active");
    properties.add("password");
    properties.add("contact");
    properties.add("shortDescription");
    properties.add("uniqueKey");
    properties.add("userType");
    return properties;
}
public UserAccount test_readUserAccount() throws Exception {
    //it is assumed that a user record exists with the unique key equal to String,
    uniqueKey
    UserAccount searchedUser = userAccountRepository.readUserAccount(uniqueKey,
    getProperties());
    return searchedUser;
}
```

1.5.5.12.4 Searching user accounts

The following sample searches users by given search criteria and returns the specified properties of the resulting user account records.

Code Snippet

```
protected UserAccountRepository userAccountRepository;
private List<UserAccount> test_readUserAccountsByCriteria() throws
Exception {
    // Criterion for user account searching
    StringFieldCriterion fieldCriterion = new StringFieldCriterion();
    fieldCriterion.setComparator(StringComparator.EQUALS_ENFORCE_CASE);
    LegacySearchFieldPathExpression fieldPathExpression = new
    LegacySearchFieldPathExpression();
    fieldPathExpression.setSearchKeyPath("username");
    fieldCriterion.setFieldPath(fieldPathExpression);
    fieldCriterion.getValue().add("JohnDoe");
    FieldSearchClause searchCriteria = new FieldSearchClause();
    searchCriteria.setOperator(LogicOperator.AND);
    searchCriteria.getCriteria().add(fieldCriterion);
    users = userAccountRepository.readUserAccountsByCriteria(searchCriteria, 100,
    getPropertiesToRead());
}
private List<String> getProperties() {
    List<String> properties = new List<String>();
    properties.add("username");
    properties.add("active");
```

```
    properties.add("password");
    properties.add("contact");
    properties.add("shortDescription");
    properties.add("uniqueKey");
    properties.add("userType");
    return properties;
}
```

1.5.5.12.5 Deleting user accounts

The following sample deletes a user account record with a given unique key.

Code Snippet

```
//for this sample it is assumed that a user record exists with a unique key String
equal to uniqueKey
protected UserAccountRepository userAccountRepository;
public void test_deleteUserAccount() throws Exception {
    userAccountRepository.deleteUserAccount(uniqueKey);
}
```

1.5.6 Glossary

A

authentication

In the TeamConnect Web Service environment, authentication is the process by which a client program establishes trusted status with the TeamConnect system (database). This is accomplished by providing a valid TeamConnect user name and password in the security header section of each Web Service request.

For more information, see SOAP Message Security Header.

H

HTTP (HyperText Transfer Protocol)

Network communications protocol used between Web Service clients and servers. In the TeamConnect Web Service environment, a developer programs in his or her desired programming language. Then he or she uses a 3rd party SOAP toolkit to transform the client program into packaged Web Service requests in SOAP messages transferred over HTTP.

O

object

In the TeamConnect Web Service environment, an object may refer to the following: an instance of a class provided in the Web Service client API, a TeamConnect system object or custom object.

R

record

In the TeamConnect Web Service environment, a record refers to an instance of a TeamConnect system object or custom object in the TeamConnect database.

repository

In the TeamConnect Web Service environment, each entity has a corresponding repository class that supports operations for working with records of that entity type. For example, the ContactRepository provides methods for inserting a contact, updating a contact, reading a contact, searching contacts, deleting a contact from a TeamConnect system.

request

In the TeamConnect Web Service environment, each client program's repository method call will be transformed by the 3rd party SOAP toolkit into a request to the corresponding Web Service.

S

SOAP (Simple Object Access Protocol)

Protocol for exchanging Web Service requests (from the client to server) and responses (from the server to the client). In the TeamConnect Web Service environment, a 3rd party SOAP toolkit will transform the client program to corresponding SOAP messages. On the server, a corresponding framework will transform the SOAP message to the appropriate programming language (and transform server responses from a programming language to a SOAP message).

Note: Mitratech does not provide a 3rd party SOAP toolkit. You need to download one from the internet.

W

Web Service

Web application programming interface (API) hosted on your TeamConnect server. In the TeamConnect Web Service environment, the Web Service API supports most operations that a TeamConnect end-user could perform in the user interface. Some operations that a TeamConnect administrator could perform in the user interface are also supported.

X

XML (extensible markup language)

In the TeamConnect Web Service environment, source files (.WSDL, .XSD) use XML format. In addition, generated SOAP messages use XML format. Generally developers do not need to create files directly in XML format. Developers should be expected to code in their desired programming language and use a 3rd party SOAP toolkit to auto-generate the corresponding SOAP (XML) messages.

Index

- 4 -

4.0.8 457

- A -

AccountRepository 1223
activateAccount 1253
allocateMoney 1254
deactivateAccount 1254
deleteAccount 1255
insertAccount 1250
method details 1250
readAccount 1251
readAccountsByCriteria 1252
readChildAccounts 1253
readRecentlyViewedAccounts 1256
transferMoney 1254
updateAccount 1251
withdrawMoney 1255
accounts object model 823
EAcctDetail 844
EAcctGroupAccess 848
EAcctUserAccess 845
JTranDetail 839
RTransaction 842
TAccount 823
actions 434
AFA rules 1044
creating pages 1048
updating 1048
appenders 71
audit 85
audit loggers 90
clearing logs 73
default 72
file layouts 77
files 74
HTML layout 80
loggers 87
pattern layout 77
rolling over logs 74
SMTP 81
socket 82
system 84
system loggers 87
viewing logs 72
XML layout 80
AppointmentRepository 1225
deleteAppointment 1259
insertAppointment 1256
method details 1256
readAppointment 1258
readAppointmentsByCriteria 1259
readRecentlyViewedAppointment 1260
updateAppointment 1258
appointments object model 804
EApptDetail 816
EApptGroupAccess 820
EApptUserAccess 818
JApptAttendee 812
JApptResource 814
LApptArealtem 811
LApptResourceType 815
TAppointment 804
approval 434
approval rules 399
approvers 434
assignee roles 139
adding 139
deleting 142
editing 141
audit rules 402
default description 404
history 405
tracking 405
triggers 403
updates 403

- B -

Backward Compatibility Settings 30
blocks 210
clearing cache 239
creating 211
creating XML files 214
defining 226
reports 220
sample XML file 223
tags 215
troubleshooting 230

blocks 210
uploading XML files 226

- C -

categories 152
activating 162
adding 159
custom object 158
deleting 162
editing 159
end-user interface 156
inactivating 160
system object 158
tree position 156
using 153
viewing 153
categories for API 1135
adding categories 1136
checking for categories 1138
default categories 1138
deleting categories 1136
returning categories 1137
clearing cache 239
client application components 1333
client proxy 1331
security headers 1332
common API functions 1145
current user 1149
system user 1147
common object model 953
JNote 953
UGrupMember 954
WObjdDetailField 955
WObjdProjectInfo 959
YDetailLookupItem 964
YDetailLookupTable 966
YGroup 967
YRecentlyViewedRecord 970
YUser 971
YUserSubscribedCollection 977
conditional expressions 475
conditions 475
creating 476
logic 484
operators 478
right arguments 478
routes 484
contact sweeper 47
custom tools 69
duplicates 54
editing 50
fixing 52
generating 48
logging 54
prerequisites 47
references 53
uploading 52
contact-centric 148
ContactRepository 1227
deleteContact 1264
insertContact 1260
method details 1260
readContact 1262
readContactsByCriteria 1263
ReadRecentlyViewedContacts 1264
updateContact 1262
contacts object model 752, 794
EContDetail 798
EContGroupAccess 802
EContUserAccess 799
JContAddress 770
JContDefaultRate 786
JContEmail 779
JContFax 777
JContInetAddress 781
JContPhone 774
JContRate 787
JContRelation 789
JContSkill 783
JContTerritory 792
JDistMember 796
LContAddressType 773
LContEmailType 780
LContFaxType 778
LContInetAddressType 782
LContPhoneType 776
LContRelationType 791
LContSkillType 784
LContTerritoryType 793
LCountryItem 772
TContact 752
converting EasyDocs RTF files to templates 631
creating EasyDocs RTF file 592
auto-mapping system fields 601
filter@ merge fields 606

creating EasyDocs RTF file 592
if@ merge fields 607
inserting filter@ merge fields 606
inserting loop@ merge fields 605
loop@ merge fields 603
mapped custom fields 602
mapped system fields 599
merge field codes 599
merge fields 597
not using filter@ 606
not using loop@ 605
creating wizards 358
actions 372
automated actions 373
General tab 360
initial actions 374
Page Components tab 364
page order 389
page transition rules 385
Page Transitions tab 382
Pages tab 362
parameters 369
simple actions 375
Wizards tab 359
CSM
admin setup 1042
designers 1038
first synchronization 1041
invoices 1041
new vendors 1041
prerequisites 1038
process setup 1040
restoring data 1052
setup 1038
uninstalling 1050
validation rules 1042
CSM rules 1048
AFAs 1044
invoices 1042
rate request approval 1043
setting screen 1043
custom action rules 400
custom blocks 1092
sample 1094
custom code 485
rule component files 485
rules 488
start up classes 487
wizard page actions 491
wizard page transitions 490
custom designs
Design Import Tool 499
Design Snapshot Tool 497
migrating 497
requirements 497
troubleshooting 516
custom field types 191
custom objects 195
Involved 194
lists 194
custom fields 182
creating 208
Data Warehouse requirements 187
name requirements 187
required fields 190
tab 185
types 191
where to use 183
custom fields for API 1138
getting values 1140
requirements 1139
setting values 1140
custom hover 249
custom hover over 249
custom lookup tables 173
activating 179
adding 175
deleting 176
deleting items 180
editing 177
inactivating 177
items 176
viewing 174
custom object fields 195
data types 201
Object Navigator 204
qualifier syntax 201
qualifiers 199
search views 198
custom objects 93
assignee roles 139
creating 110
deleting 152
embedded 100
general information 112
Involved 93

custom objects 93
naming patterns 122
notifications 143
phase transitions 134
phases 130
unique IDs 117
custom pages 180
blocks 210
custom fields 182
object views 231
process 182
views 181
custom pages in API 1091
blocks 1092
screens 1092
tools 1098
custom scheduled actions 1101
adding to TeamConnect 1101
sample 1102
custom screens
API sample 1094
custom screens in API 1092
custom tools 69
API sample 1099
configuring 70
in API 1098
uploading 70
custom views 181, 249
customization 26
custom messages 35
Data Warehouse 36
design 26
sequence 32
Setup 27
SQL Server 36
system settings 37
user interface 27

- D -

design 26
Design Import Tool 499
design upgrades 500
Design Snapshot Tool 497
creating snapshot 498
reverting 499
upgrade file 499
Document Generator 520
content 541
headers 526
user input 538
document template tags 544
tc:block 567
tc:compute 568
tc:conditional 562
tc:data 547
tc:date 560
tc:detail 552
tc:file 561
tc:filter 572
tc:input 575
tc:loop 554
tc:operand 569
tc:search 570
tc:user 560
document templates
basics 521
completing 584
content 541
converting templates to XML 585
creating headers 585
creating tags 584
creating user input fields 538
displaying entry fields 540
Document Generator 520
document templates 525
filter pages 526
filtering sub-objects 650
headers 526
main filters 534
nested tags 576
nesting tags 641
non-related object filters 533
object attribute names 579
record filters 530
related object filters 529
retrieving data from child objects 647
retrieving data from contact fields 643
retrieving data from custom fields 646
retrieving data from Involved custom fields 647
retrieving data from related objects 645
samples 641
sub-filters 534
sub-object filters 530
tags 544

document templates
tc:conditional samples 651
testing 587
troubleshooting 635
uploading 587
user input 538
writing text 542
DocumentRepository 1233
checkIn 1274
checkOut 1274
copyDocument 1276
createHyperlink 1268
createHyperlinkDefaultCategory 1268
createShortcut 1267
createSubFolder 1267
deleteDocument 1277
insertDocument 1265
method details 1264
moveDocument 1276
raedChildDocuments 1272
readChildDocumentForName 1272
readDocument 1270
readDocumentFolderForDocumentOwner 1273
readDocumentsByCriteria 1270
readDocumentsByPath 1271
readFolderHierarchy 1278
ReadRecentlyViewedDocuments 1277
revet 1275
setDocumentAsRecentlyViewed 1277
undoCheckOut 1275
updateDocument 1269
documents object model 937
EDocuDetail 947
EDocuGroupAccess 951
EDocuUserAccess 948
JDocuContent 945
TDocument 937
YDocuContentType 945
documents templates
inserting tags 580
preparing RTF files 580

- E -

EAcctDetail 844
EAcctGroupAccess 848
EAcctUserAccess 845
EApptDetail 816
EApptGroupAccess 820
EApptUserAccess 818
EasyDocs 588
converting RTF files to templates 631
creating RTF file 592
Data Mapping Tool 608
deleting template files 635
Document Templates folder 608
functions 589
group rights 592
mapping RTF merge fields 613
modifying data mapping 634
modifying RTF files 633
process 590
step 1 592
step 2 612
step 3 613
step 4 631
step 5 632
template files 633
testing templates 632
uploading RTF file 612
EContDetail 798
EContGroupAccess 802
EContUserAccess 799
Editing items pending approval 457
EDocuDetail 947
EDocuGroupAccess 951
EDocuUserAccess 948
EExpeDetail 856
EExpeGroupAccess 860
EExpeUserAccess 858
EHistDetail 930
EHistGroupAccess 935
EHistUserAccess 933
EInvcDetail 913
EInvcGroupAccess 923
EInvcUserAccess 920
EInvlDetail 730
EInvlGroupAccess 734
EInvlUserAccess 732
ELitmDetail 914
ELitmGroupAccess 916
ELitmUserAccess 918
embedded objects 100, 148
converting to child 149
creating 145

embedded objects 100, 148
defining 148
displaying 144
list displays 143
EMileDetail 745
EMileGroupAccess 749
EMileUserAccess 746
enumerations 1074
EProjDetail 716
EProjGroupAccess 720
EProjUserAccess 717
ETaskDetail 873
ETaskGroupAccess 877
ETaskUserAccess 874
ExpenseRepository 1236
deleteExpense 1282
insertExpense 1279
method details 1278
postExpense 1281
readExpense 1280
readExpensesByCriteria 1281
readRecentlyViewedExpenses 1282
updateExpense 1279
voidExpense 1282
expenses object model 850
EExpeDetail 856
EExpeGroupAccess 860
EExpeUserAccess 858
TExpense 851
Export List 503

- F -

field value types 341
memo 341
number 341
text 341
file names 997
base classes 1000
batch display files 1000
custom Java blocks 1000
parts 997
utility classes 999
frequently asked questions
CSM 1067
web services 1329

- G -

global navigation 327
Tab bar 328
Tab bar contents 329
GroupAccountRepository 1237
deleteGroupAccount 1286
insertGroupAccount 1283
method details 1283
readGroupAccount 1284
readGroupAccountsByCriteria 1285
readRecentlyViewedGroupAccounts 1286
updateGroupAccount 1284
groups
preparing 26

- H -

histories object model 925
EHistDtail 930
EHistGroupAccess 935
EHistUserAccess 933
THistory 926
HistoryRepository 1238
deleteHistory 1290
insertHistory 1287
method details 1287
readHistoriesByCriteria 1289
readHistory 1288
readRecentlyViewedHistories 1290
updateHistory 1288
home pages
adding portal panes 325
basics 285
content 311
Content tab 323
creating 312
customization 309
customizing 314
editing 317
layout 317
master 315
modifying 316
settings 310
summary 284
synchronizing customizations 317
hover over 249

- I -

invoice validation rules 1042
adjusting 1054
do not allow multi-matter invoices 1056
expense charges exceed acceptable unit costs 1055
fee charges exceed agreed rates 1054
invoice contains duplicate expense 1063
invoice contains duplicate timekeeper charge 1061
invoice currency doesn't match vendor currency 1066
invoice is too old 1060
invoice line item description contains unauthorized keywords 1065
invoice service/expense charges are too old 1059
timekeeper billed too much time 1057
InvoiceRepository 1239
adjustInvoiceHeader 1296
deleteInvoice 1296
insertInvoice 1291
method details 1290
postInvoice 1295
readActiveApprovals 1297
readCompletedApprovals 1298
readInvoice 1293
readInvoiceApprovalsPendingOnPost 1298
readInvoicesByCriteria 1294
readRecentlyViewedInvoices 1296
updateInvoice 1293
voiceInvoice 1295
invoices object model 879
EInvcDetail 913
EInvcGroupAccess 923
EInvcUserAccess 920
ELitmDetail 914
ELitmGroupAccess 916
ELitmUserAccess 918
JInvcCategoryAdjustment 893
JInvcHeaderAdjustment 895
JInvcLineItem 898
JInvcNonUSTax 906
JInvcTimekeeperAdjustment 909
JLitmAdjustment 910
LInvcNonUSTaxType 907
TInvoice 879
YCurrencyInfo 924
Involved objects 93
creating 115
InvolvedRepository 1246
deleteInvolved 1311
insertInvolved 1307
method details 1307
readInvolved 1309
readInvolvedsByCriteria 1310
readInvolvedsForProject 1309
updateInvolved 1308

- J -

JApptAttendee 812
JApptResource 814
Java samples 1169
automated actions 1179
automated qualifiers 1171
categories 1169
child projects 1169
parameterized actions 1187
related records 1169
rule actions 1179
searching 1197
Wizard page actions 1192
JContAddresss 770
JContDefaultRate 786
JContEmail 779
JContFax 777
JContInetAddress 781
JContPhone 774
JContRate 787
JContRelation 789
JContSkill 783
JContTerritory 792
JDistMember 796
JDocuContent 945
JInvcCategoryAdjustment 893
JInvcHeaderAdjustment 895
JInvcLine Item 898
JInvcNonUSTax 906
JInvcTimekeeperAdjustment 909
JInvRelation 729
JLitmAdjustment 910
JNote 953

JProjAssignee 704
JProjPhase 708
JProjRelation 713
JTaskAssignee 872
JTranDetail 839

- L -

LApptArealtem 811
LApptResourceType 815
LContAddressType 773
LContEmailType 780
LContFaxType 778
LContInetAddressType 782
LContPhoneType 776
LContRelationType 791
LContSkillType 784
LContTerritoryType 793
LCountryItem 772
LInvcNonUSTaxType 907
list fields for API 1140
  getting values 1145
  setting values 1145
localization
  custom blocks 494
  custom messages 494
  customizing 491
  i18n keys 492
  portal panes 494
  reports 493
  rules 494
  saved searches 495
  Unique Key fields 493
  upgrade considerations 495
  wizards 495
logging using API 1166
  logger levels 1168
  logging messages 1167
lookup tables 152
  activating 172
  adding 169
  custom 173
  deleting 173
  editing 170
  inactivating 170
  system 164
  types 163
  using 163
lookup tables for API 1140
  custom lookup table tree positions 1144
  system lookup table tree positions 1142
LookupTableSource 1243
  method details 1299
  readSystemLookupTable 1299
LProjAssigneeType 706
LProjRelationType 714
LTaskActivityItem 870

- M -

mapping EasyDocs RTF merge fields 613
  converting to XML files 630
  current date 628
  current user 628
  custom field mapping 617
  filter@ merge field 621
  filter@ to related objects 622
  filter@ to sub-objects 622
  if@ merge field mapping 625
  loop@ merge field 618
  loop@ to related objects 618
  loop@ to sub-objects 618
  nested within filter@ 623
  nested within loop@ 619
  opening RTF file 613
  system field mapping 614
  to a file 629
  to XML files 630
  user input screens 631
matter management
  adding CSC mapping files 1036
  adding CT mapping files 1035
  assignee roles 1005
  categories 1003
  CSC mapping files 1036
  CSM configuration 1028
  CT mapping files 1035
  custom fields 1006
  custom screens 1008
  custom tools 1017
  customizing 996
  E-Billing Roles tab 1029
  embedded objects 1002
  groups 1030
  lookup tables 1007

matter management
object definitions 1001
payment settings 1030
phase transitions 1004
phases 1004
reports 1011
rules 1009
SOP 1034
Time Entry Tool 1018
tool rights 1027
users 1030
vendor settings 1029
wizards 1012
model classes 1072

- N -

nesting tags 641

- O -

object definitions 102
custom objects 110
system objects 108
object model 681
accounts 823
appointments 804
audience 682
columns 685
common objects 953
contacts 752
conventions 683
database table names 691
documents 937
enumerations 683
expenses 850
histories 925
invoices 879
legacy names 682
list attributes 691
lookup tables 689
new names 682
object names 691
objects 682
projects 692
properties 687
system objects 689
table prefixes 687
tasks 862
terms 684
XCT files 691
object model properties 687
database table names 691
list attributes 691
lookup tables 689
object names 691
system objects 689
table prefixes 687
XCT files 691
object naming patterns 122
attribute format 126
auto naming 130
literals 128
object attributes 125
object navigator 665
attributes 666
creating paths 676
filtering related objects 673
filtering sub-objects 673
navigating objects 678
paths 668
starting tables 671
window 668
object views 231
assigning 238
clearing cache 239
creating 236
defining 235
invoice line items 240
troubleshooting 238
objects 91
creating 101
custom 93
default icons 109
definitions 102
embedded 100
Involved 93
related 93
sub-objects 99
system 91
objects definitions
viewing 105

- P -

phase transitions 134

phase transitions 134
creating 137
deleting 138
editing 137
phases 130
adding 132
deleting 133
editing 132
portal panes 287
action links 296
basics 285
changes 318
color 320
contents 320
creating 308
creating from home page 326
custom content 305
customizing 314
default 287
deleting 321
modifying 316
multiple contents 307
object links 297
opening 288
options 320
search views 301
settings 289
summary 284
synchronizing customizations 317
templates 315
text 302
title 320
Web URLs 304
WebIntelligence URLs 306
portal panes contents 292
post commit rules 406
pre-population rules 398
ProjectRepository 1243
changePhase 1304
deleteProject 1305
insertProject 1300
method details 1300
readChildProjectsForEntityType 1303
readProject 1302
readProjectEntityTypes 1305
readProjectIntegrationSearches 1306
readProjectsByCriteria 1302
readProjectsUsingSearch 1306
readRecentlyViewedProjects 1305
updateProject 1301
projects object model 692
EInvlDetail 730
EInvlGroupAccess 734
EInvlUserAccess 732
EMileDetail 745
EMileGroupAccess 749
EMileUserAccess 746
EProjDetail 716
EProjGroupAccess 720
EProjUserAccess 717
JInvlRelation 729
JProjAssignee 704
JProjPhase 708
JProjRelation 713
LProjAssigneeType 706
LProjRelationType 714
TInvolved 722
TMilestone 737
TProject 692
WObjdPhaseTransition 712
WObjdPhaseType 709

- Q -

qualifier items 419
creating 427
examples 428
limitations 431
operators 421
right arguments 423

- R -

rate request approval rule 1043
reassign 457
related objects 93
contact-centric 97
parent-child relationships 96
relationship types 94
related records for API 1131
custom fields 1132
getting 1132
primary keys 1132
REST API 1321
retrieving tc:loop data 554
qualifier examples 559

retrieving tc:loop data 554
qualifier syntax 557
qualifiers 556
sub-objects 555
retrieving template data
from child objects 647
from contact fields 643
from custom fields 646
from Involved custom fields 647
from related objects 645
routes 432
accessing 458
conditions 484
customizing 469
email notifications 471
General tab 459
stops 459
RTransaction 842
rule actions 1082
custom code 485
parameters 1087
writing 1084
writing in Java 1085
rule qualifers 1082
custom code 485
parameters 1087
writing 1083
writing in Java 1085
rule triggers 406
events 406
rule types 396
approval 399
audit 402
custom action 400
post commit 406
pre-population 398
scheduled action 401
security 397
user invoked action 401
validation 398
rules 395, 434
approval 399
audit 402
automated actions 1082
automated qualifiers 1082
comoponents 396
creating 410
creating qualifiers 427
custom action 400
general information 410
General tab 414
post commit 406
pre-population 398
qualifier items 419
qualifier logic 425
Qualifier tab 418
qualifiers 417
scheduled action 401
security 397
specifications 473
tab 410
trigger events 406
triggers 406
types 396
user invoked action 401
validation 398
rules in Java 1085
automated qualifier 1089

- S -

scheduled action rules 401
seach views
creating 246
search view filter 273
contacts 278
creating 279
filter tab 277
qualifier conditions 274
qualifier tips 281
qualifiers 273
section titles 280
search view results 266
object link types 271
object links 272
search example 270
spanned columns 270
search views 249
approvals 253
auto search 261
designing 242
document content 254
field content 255
filter display 273
full-text search 254
general information 256

search views 249
  global search 250
  home pages 247
  IMAP search 263
  projects 265
  requests 253
  results display 266
  screen 243
  search modules 262
  subscribing 260
  templates 266
  Third-party calendars 265
  troubleshooting 284
  unsubscribing 260
  workflow processes 253
searching using API 1155
  executing 1165
  field paths 1162
  on fields 1161
  search criteria 1157
  search operators 1164
  search parameters 1164
  search views 1157
security rights for API 1150
  functional security 1151
  record security 1150
  system user 1147
security rules 397
service classes 1074
  actions 1077
  dependencies 1078
  other 1078
  qualifiers 1077
  ResourceService 1075
  screens 1077
Setup 27
  Backward Compatibility Settings page 30
  designer menu 28
  overview 27
ssearch view results
  display columns 272
stop members 463
  groups 464
  user paths 465
  users with roles 466
stops 459
  customizing 469
sub-objects 99
  embedded 100
sub-objects for API 1133
  contacts 1135
  project assignees 1134
  task assignees 1134
system lookup tables 164
  contact address 168
  structure 167
system objects 91
  configuring 108
  default icons 109
  general information 108
system settings 37
  accessing 37
  appenders 71
  contact sweeper 47
  duplicate checking 55
  duplicate contact manager 46
  invoice duplicates 41
  invoices 40
  line items 41
  line items behavior 41
  miscellaneous 38
  non-US taxes 40
  preventing duplicates 55
  rule execution 37
  XML Worksheet tool 68
system views 181
sytem settings
  line item adjustments 43

- T -

Tab bar
  appearance 328
  contents 329
TAccount 823
tag attributes 654
  all field types 656
  custom object fields 664
  date fields 662
  field labels 655
  Involved fields 664
  list fields 662
  memo text fields 661
  number fields 658
  text fields 660
TAppointment 804

TaskRepository 1247
deleteTask 1315
insertTask 1311
method details 1311
postTask 1314
readRecentlyViewedTasks 1315
readTask 1313
readTasks 1313
reassign 1315
updateTask 1312
voidTask 1314
tasks object model 862
ETaskDetail 873
ETaskGroupAccess 877
ETaskUserAccess 874
JTaskAssignee 872
LTaskActivityItem 870
TTask 862
tc:conditional 562
categories 567
comparing custom field values 565
comparing sub-objects 566
custom fields 651
meeting conditions 653
object attribute values 566
samples 651
sub-objects 652
testing 652
TContact 752
TDocument 937
TeamConnect API 1070
account samples 1105
accounts 1105
actions 1082
appointment samples 1107
appointments 1107
blocks 1092
categories 1135
common functions 1145
contact samples 1110
contacts 1109
custom fields 1138
custom objects 1128
custom settings 1153
development environment 1071
document samples 1112
documents 1112
enumerations 1074
executing search 1165
expense samples 1115
expenses 1115
getting started 1078
group samples 1117
groups 1117
history 1119
history samples 1119
internationalization 1154
invoice samples 1121
invoices 1120
Involved parties 1130
Involved party samples 1130
Java samples 1169
list fields 1140
logging 1166
lookup tables 1140
model classes 1072
new records 1103
object definitions 1103
objects 1103
packages 1072
pages 1091
policy 1069
project samples 1129
projects 1128
qualifiers 1082
related records 1131
ResourceService 1075
rules 1082
scheduled actions 1101
screens 1092
search criteria 1157
search parameters 1164
searching 1155
security rights 1150
service classes 1074
sub-objects 1133
system objects 1105
system settings 1152
task samples 1123
tasks 1123
tools 1098
user samples 1126
user settings 1152
users 1126
TeamConnect customization 26
TeamConnect Enterprise

TeamConnect Enterprise
API 1068
designers 25
developers 1068
web services 1200
templates 330
adding fields 338
assignees 346
attendees 346
attribute values 343
categories 345
creating 336
defining fields 338
deleting sub-objects 348
document folders 334
features 333
field values 341
formula values 343
general information 333
General tab 333
literal values 342
planning 331
Records tab 335
related objects 348
sub-objects 344
testing EasyDocs templates 632
TExpense 851
THistory 926
Time Entry Tool Settings 1018
administrator 1025
default fields 1020
editing 1018
time periods 1022
timekeepers 1023
TInvoice 879
TInvolved 722
TMilestone 737
TProject 692
troubleshooting
CSM 1067
troubleshooting custom designs 516
BUILD FAILED message 520
design import restrictions 517
DMT script errors 520
error messages 518
general steps 518
logging 518
no design components 517
performance issues 520
warnings 516
TTask 862

- U -

UGrupMember 954
unique IDs 117
auto numbering patterns 119
object attributes 121
project link display 121
upgrade considerations for localization 495
best practices 496
custom blocks 495
date formatting 496
parameters 496
uploading EasyDocs RTF file 612
user involved action rules 401
UserAccountRepository 1249
deleteUserAccount 1320
insertUserAccount 1316
method details 1316
readRecentlyViewedUserAccounts 1320
readUserAccount 1318
readUserAccountsByCriteria 1319
updateUserAccount 1318
users
preparing 26

- V -

validation rules 398

- W -

web service API reference 1222
AccountRepository 1223
AppointmentRepository 1225
ContactRepository 1227
details 1250
DocumentRepository 1233
ExpenseRepository 1236
GroupAccountRepository 1237
HistoryRepository 1238
InvoiceRepository 1239
InvolvedRepository 1246
LookupTableSource 1243

web service API reference 1222
ProjectRepository 1243
summary 1222
TaskRepository 1247
UserAccountRepository 1249
web service applications 1213
categories 1218
client proxy 1213
common functions 1215
components 1215
creating records 1216
custom fields 1220
deleting records 1221
reading records 1216
searching records 1216
SOAP security header 1214
updating records 1216
URLs 1213
web service code samples 1335
accounts 1343
appointments 1350
contacts 1336
documents 1354
expenses 1366
group accounts 1369
history 1373
invoices 1376
Involved parties 1390
line items 1376
projects 1385
tasks 1394
user accounts 1398
web services 1200
abstract classes 1324
API reference 1222
batch operations 1329
category data objects 1326
client application components 1330
client source-code 1209
code samples 1335
cosmetic data objects 1330
creating applications 1213
custom field data objects 1326
data types 1207
error handling 1208
getting started 1200
glossary 1401
key concepts 1206
non-supported operations 1329
overview 1200
record reading 1207
record updating 1207
repository classes 1206
request formats 1207
requirements 1208
response formats 1207
REST API 1321
search criteria data objects 1327
supported operations 1202
unique keys 1208
web services for accounts 1343
activating 1348
allocating money 1349
creating 1343
deactivating 1349
deleting 1350
reading 1345
reading child accounts 1346
searching 1347
transferring money 1349
updating 1344
withdrawing money 1350
web services for appointments 1350
creating 1350
deleting 1354
reading 1352
searching 1353
updating 1351
web services for contacts 1336
creating 1336
deleting 1342
reading 1340
searching 1340
updating 1338
web services for documents 1354
checking in 1365
checking out 1365
copying 1364
creating 1354
creating hyperlinks 1364
creating hyperlinks with default category 1363
creating shorcuts 1363
creating subfolders 1363
deleting 1366
moving 1364
reading 1356

web services for documents 1354
reading by path 1358
reading child documents 1357
reading documents of parent folder 1359
reading folder for user 1360
reverting to previous version 1365
searching 1361
undoing checkout 1365
updating 1355
web services for expenses 1366
creating 1366
deleting 1369
posting 1369
reading 1367
searching 1368
updating 1366
voiding 1369
web services for group accounts 1369
creating 1370
deleting 1372
reading 1371
searching 1372
updating 1370
web services for history 1373
creating 1373
deleting 1376
reading 1374
searching 1375
updating 1373
web services for invoices 1376
adjusting 1378
creating 1376
deleting 1385
posting 1383
reading 1380
searching 1381
updating 1378
voiding 1383
web services for Involved parties 1390
deleting 1394
inserting 1390
reading 1392
reading for projects 1392
searching 1393
updating 1391
web services for line items 1376
adjusting 1379
creating 1376
searching 1383
web services for projects 1385
changing phases 1388
creating 1385
deleting 1390
reading 1388
reading child projects 1389
searching 1389
updating with embedded records 1387
web services for tasks 1394
creating 1394
deleting 1398
posting 1397
reading 1395
reassigning 1398
searching 1396
updating 1395
voiding 1397
web services for user accounts 1398
creating 1398
deleting 1401
reading 1399
searching 1400
updating 1399
wizards 351
actions 372
automated actions 373
creating 358
designing 351
execution 395
General tab 360
initial actions 374
layout 356
Page Components tab 364
page order 389
page transition rules 385
page transitions 357
Page Transitions tab 382
pages 362
parameters 369
requirements 353
simple actions 375
specifications 353
starting 395
tab 359
templates 355
testing 390
troubleshooting 390

Wizards tab 359
WObjdPhaseTransition 712
WObjdPhaseType 709
WObjdProjectInfo 959
WObjfDetailField 955
workflow 434, 457

- Y -

ycontactgroup
Y_Contact_Group 794
YCurrencyInfo 924
YDetailLookupItem 964
YDetailLookupTable 966
YDocuContentType 945
YGroup 967
YRecentlyViewedRecord 970
YUser 971