



# StarTeam 17.0

Eclipse Plugin

**Micro Focus**  
**The Lawn**  
**22-30 Old Bath Road**  
**Newbury, Berkshire RG14 1QN**  
**UK**  
<http://www.microfocus.com>

© Copyright 2019 Micro Focus or one of its affiliates.

**MICRO FOCUS**, the Micro Focus logo and StarTeam are trademarks or registered trademarks of Micro Focus or one of its affiliates.

All other marks are the property of their respective owners.

2019-01-27

# Contents

<b>Achieving Consistent Check-ins and Check-outs</b>	<b>7</b>
<b>StarTeam Eclipse Plugin</b>	<b>8</b>
Introduction	8
Installation and Licensing for StarTeam	8
Products Included with StarTeam Enterprise Licenses	9
Products Included with StarTeam Enterprise Advantage Licenses	11
About Source Control	12
Standard StarTeam Architecture Overview	12
Contacting Support	14
StarTeam Eclipse Plugin	14
Getting Started	14
Logging On	14
Logging Off	15
Tour of the UI	15
Sharing Workspace Projects	29
Create Projects and Synchronize Local Files	30
Checking Out Existing Folders	30
Setting Preferences	31
Opening Perspectives and Views	31
Configuring Capabilities	31
Creating a New Connection	31
Removing Connections	32
Configuring Profiles	32
Deleting Server Configurations	32
Linking Views with Selections	32
Exporting a Set of Team Preferences	33
Importing a Set of Team Preferences	33
Displaying Additional Fields	33
Setting Background Operations	34
Viewing Background Operations	34
Refreshing Server Configurations	34
Compare Revisions	34
StarTeam Basics	34
Containers	35
Artifacts	36
Artifacts Versus Items	37
Folders	38
Folders and Views	39
Files	40
Change Requests	41
Requirements	46
Tasks	47
Topics	47
Links: Internal and External	48
Labels	48
Branching, Merging and Dot Notation	51
Sharing and Cheap Copies	54
Promotion States	56
Audit Log	56
Table of Common Operations	60
User-Defined Property Fields	63

Displaying Custom Property Fields .....	63
Creating Content Fields .....	63
Creating Date and Time Fields .....	64
Creating Enumerated Fields .....	65
Creating Group and Group List Fields .....	66
Creating Map Fields .....	66
Creating Rich Content Fields .....	67
Creating Text Fields .....	67
Creating Time Span Fields .....	68
Creating User and User List Fields .....	69
Projects .....	69
Project Structure .....	70
Guidelines for Keeping Projects Autonomous .....	71
Cross-Project Activity Support .....	72
How to Handle Cross-Project File Dependencies .....	73
Working with Projects .....	74
Project Properties .....	85
View Configuration and Management .....	89
Overview of Views .....	89
View Types .....	91
View Roles .....	93
Proper Use of Views .....	96
View Configuration Options .....	96
View Type Options and Settings .....	97
Working with Views .....	100
Branching .....	106
References Overview .....	108
Understanding References .....	108
References Created by Branching Views .....	111
References Created by Adding Items to Views .....	113
References Created by Manually Sharing Objects .....	115
References Created by Moving Objects .....	116
Personal Preferences .....	118
Set of Team Preferences .....	118
Setting Preferences .....	118
StarTeam Preferences .....	118
Folders .....	139
Folders and Items .....	139
Working with Folders and Items .....	157
Links: Internal and External .....	169
Linking Items Internally or Externally .....	169
Deleting Links .....	170
Linking Files to Process Items .....	170
Linking Specific Revisions .....	171
Customizing Link Item Properties .....	172
Customizing Link Properties .....	172
Finding Files Associated with Active Process Items .....	173
Reviewing Linked Change Requests .....	173
Change Requests .....	173
Change Request View .....	174
Creating Change Requests .....	176
Editing Change Requests .....	178
Viewing Unread Change Requests .....	178
Default and Required Change Request Fields .....	178
Moving Change Requests .....	180
Assigning Change Requests .....	180

Resolve Open Change Requests	180
Verifying Resolved Change Requests	181
Closing Verified Change Requests	182
Reviewing Linked Change Requests	182
Change Request Lifecycle	182
Change Request Status Field	183
Change Request Properties	183
Change Request Fields	184
Commonly Used Change Request Abbreviations	192
Sample Change Request Detail View Template	192
Requirements	194
Requirement View	195
Creating Requirements	196
Sample Requirement Detail View Template	197
Requirement Properties	198
Requirement Fields	199
Tasks	206
Creating Tasks	206
Assigning Task Resources	207
Estimating Tasks	207
Adding Notes to Tasks	208
Working with Work Records in Tasks	208
Sample Task Detail View Template	209
Task Fields	210
Task Properties	217
Topics	219
Creating Topics	220
Responding to Topics or Responses	221
Sample Topic Detail View Template	222
Topic Fields	223
Topic Properties	227
Files	228
Opening Files	230
Modifying File Properties	230
Enabling Concurrent File Editing	230
Excluding Files from a Project	231
Finding Files Associated with Active Process Items	231
Marking Unlocked Files Read-only	232
Setting the File Executable Bit for UNIX	232
Viewing Annotations	232
Viewing Previous File Revisions	233
Comparing and Merging Files	233
Converting Resource Change Status	233
Editing Check-in Comments	234
Enabling Quick Diff in Text Editors	234
Force Check-out with Keyword Expansion	234
Locking and Unlocking Items	234
Modifying the EOL Conversion Mode	235
Viewing or Modifying Item Properties	235
Sample File Detail View Template	235
File Properties	236
File Fields	237
Check-in and Check-out Operations	245
Atomic Check-ins	246
Check-in and Check-out Overview	246
EOL Conversion Handling Overview	251

Process Items and Process Rules .....	253
Process Items .....	253
Process Items and Workspace Change Packages .....	253
Process Rules .....	254
Active Process Items .....	255
Promoting File Changes Into Baselines .....	256
Displaying Only Enhanced Process Links .....	256
Filtering Process Tasks From Other Tasks .....	256
Setting Active Process Items .....	257
Establishing Process Rules for Projects .....	257
File Compare/Merge .....	258
Main File Compare/Merge .....	258
Embedded File Compare/Merge .....	259
Standalone File Compare/Merge .....	259
Specifying FCM as Alternate Compare/Merge Tool .....	259
Labels and Promotion States .....	259
Labels .....	260
View Labels .....	262
Creating View Labels .....	263
Creating Revision Labels .....	264
Configuring or Viewing Label Properties .....	264
Attaching Labels to Folders .....	265
Attaching Labels to Items .....	266
Demoting View Labels .....	267
Promoting View Labels .....	267
Copying Revision Labels .....	267
Deleting Labels .....	268
Detaching a Label from a Specific Revision .....	268
Detaching Labels from Folders .....	268
Detaching Labels from Items .....	269
Freezing or Unfreezing Labels .....	269
Promotion States .....	270
Filters and Queries .....	273
Filtering Data .....	273
Queries .....	277
Terminology .....	283

# Achieving Consistent Check-ins and Check-outs

Developers can use various StarTeam features to allow or avoid conflicts with other developers on the same files (in the same view). In a low-contention environment, developers can check-out files without locks, modify them, “refresh” to identify and resolve merge conflicts, and then check in modified files. All check-ins in StarTeam are atomic. If more than one file is checked in as the result of a single transaction (for example, in a change package from a View Compare/Merge session) the files and their associated process items are updated in a single action. If for some reason, the check-in fails, none of the files are checked in, and the status of the associated process items is not updated. In general, you can achieve consistent check-ins and check-outs by doing the following:

- Exclusively lock files before checking them in or out and unlock files after a successfully checking them in or out; or
- Temporarily change your view configuration to a known “stable” point

In higher contention environments, developers may want more assurance of getting consistent sets of files during check-out, that is, avoiding files that are a partial set of someone else’s check-in. The easiest way to achieve this need is “by convention”. Each developer exclusively locks all files before checking them in, and unlocks them when they are “complete”, either at check-in or soon thereafter. Correspondingly, each developer exclusively locks all files before checking them out, and unlocks them when complete. If a developer cannot get all the locks at once, they are potentially about to interfere with another developer, so they unlock files they currently have locked, wait a bit (probably talk with the developer they are conflicting with), and then try again. One implication of this “by convention” approach is that a developer could be blocked while waiting for another developer to finish-up.

A more formal way to enforce consistent check-outs is to use “view configuration”. To ensure consistent check-outs without locks, a developer can temporarily change their view configuration to a known “stable” point. In some organizations, a nightly build process creates a view label when the server is not used or lightly used. To temporarily change the view configuration from the StarTeam client, select **View > Select Configuration > Labeled Configuration** and choose the latest build label. (Alternatively, choose a timestamp or promotion state.) The view will switch to show the item states at the selected time, and “consistent” check-outs can be performed from there.



**Note:** The view configuration change is only at the client – the underlying “real” view is not modified. Also, note that “rolled-back” views are read-only: they must be reset to the “current” configuration before new/modified files can be checked-in. An implication to be aware of with this approach is that the time to switch the configuration can take a few seconds to a few minutes on very large views.

# StarTeam Eclipse Plugin



Welcome to StarTeam Eclipse Plugin

[StarTeam Eclipse Plugin](#)  
[Tour of the UI](#)  
[Licensing for StarTeam](#)  
[Getting Started](#)



Online resources

[Micro Focus Infocenter](#)  
[Micro Focus SupportLine](#)  
[Micro Focus Product Updates](#)  
[Micro Focus Knowledge Base](#)  
[Micro Focus Community Forums](#)  
[Micro Focus Training Store](#)



Provide feedback

[Contacting Support](#)  
[Email us feedback regarding this Help](#)

## Introduction

This section provides introductory information about StarTeam.

## Installation and Licensing for StarTeam

### Installation

Installation instructions for installing StarTeam products can be found in the *StarTeam Installation Guide*.

### Licensing

StarTeam is available in three licensing packages:

- Enterprise** StarTeam Enterprise provides a basic feature set, including the StarTeam Server, StarTeamMPX (MPX Event Transmitter and MPX Message Broker), the Cross-Platform Client, StarTeam Web Client, LDAP QuickStart Manager, and the SDK. The requirements component is not available with this license, however, it does provide access to custom fields.
- Enterprise Advantage** StarTeam Enterprise Advantage has all the StarTeam Enterprise features plus the Requirement component, StarTeamMPX (MPX Cache Agent and MPX File Transmitter),



and StarTeam Workflow Extensions which include alternate property editors (APEs) that enable you to create custom forms and design workflow rules to control how all the items in a component move from state to state. StarTeam Datamart is available for purchase.

## Products Included with StarTeam Enterprise Licenses

The following provides a summary of StarTeam products that come with the StarTeam Enterprise license. The installation instructions for some products are not in this consolidated installation guide, but are located in the respective guide of that product and are noted.

<b>StarTeam Server</b>	A StarTeam Server stores artifacts (files, change requests/defects, tasks, and topics) for StarTeam clients. A server can support one or more server configurations on the same computer. Install StarTeam Server on a computer that is accessible to all users.
<b>MPX Message Broker</b>	Pushes information from the StarTeam Server to its clients. Usually an administrator sets up a cloud of Message Brokers to improve server performance for users in diverse geographic locations. One (sometimes two) root Message Brokers are set up for the server, usually on the same computer or in a network-near location.
<b>StarTeam Cross-Platform Client</b>	The StarTeam Cross-Platform Client is the most used client and provides users with access to all of the artifacts on the server. The Cross-Platform Client is a pure Java client that provides support of operating systems where a compatible JRE or JDK are available. As such, Cross-Platform Client is available for the Microsoft Windows, Solaris, and Linux operating systems.
<b>StarTeam Visual Studio Plugin</b>	The StarTeam Visual Studio Plugin provides the StarTeam software configuration management capabilities tightly integrated with the Microsoft Visual Studio development environment. Using this integration makes it possible for you to develop applications in the Microsoft Visual Studio environment while simultaneously using the version control, change request, topic, task, and requirement component assets of StarTeam. The integration brings StarTeam main menu commands, context menu commands, and an embedded StarTeam client (providing much of the same look-and-feel as the full-featured Cross-Platform Client) to the Microsoft Visual Studio development environment.
<b>StarTeam Eclipse Plugin</b>	StarTeam Edition for Eclipse allows you to share projects on StarTeam Server and projects in the Eclipse workspace, but it is much more than just a version control plugin. This integration offers project teams a customizable solution providing requirements, task, and change management, defect tracking and threaded discussions tightly integrated within the Eclipse platform.
<b>StarTeam Web Server and StarTeam Web Client</b>	The StarTeam Web Server makes it possible for users to access the server from their browsers using the StarTeam Web Client. The StarTeam Web Client is an intuitive web-based interface that many simultaneous users can use to connect to one or more StarTeam Servers to access projects and manage items. This product contains a core feature set designed to meet the needs of users responsible for viewing, creating, and editing StarTeam change requests, requirements, tasks, and topics. Browsing files and a limited set of file operations are also available.  <b>Note:</b> You must have a StarTeam user license to use the Web Client.
<b>LDAP Quickstart Manager</b>	The StarTeam Server can provide password authentication via a directory service, such as LDAP Quickstart Manager (QSM) to add users to the server, along with their distinguished names (DN) (needed for authentication) and other user information.
<b>Layout Designer</b>	Use Layout Designer to create forms for artifacts, such as change requests. This allows you to put the most important properties on the first tab, etc. With the web client

and an Enterprise Advantage server, a Layout Designer form works with workflow. This is not true of the StarTeam Cross-Platform Client where Layout Designer's use is only for form building.

This product is translated into English, French, German, and Japanese.

**StarTeam SDK** The StarTeam SDK is cross-compiled so that it can be offered both as Java and .NET applications. The full SDK is used by developers to create additional applications that use the StarTeam Server.

Usually, the StarTeam SDK runtime is installed with clients automatically so it can be used by them to access the StarTeam Server. Occasionally, you may need to install the runtime.

**StarTeam SCC Integration** The StarTeam SCC Integration works with any application that uses the Microsoft Source Code Control (SCC) Application Programming Interface (API). This API, originally designed by Microsoft to allow applications to work with Microsoft Visual SourceSafe, enables you to perform version control operations, such as checking files in and out, using StarTeam as the SCC provider.

**StarTeam Quality Center Synchronizer** This product is available with all licenses.

StarTeam Quality Center Synchronizer can ensure that the same data appears in Quality Center and a database used by StarTeam Server. The goal of the synchronization is to provide access to the latest information about defects, whether the defects are being processed from Quality Center or from StarTeam. You can use Quality Center to add defects, and you can use StarTeam to indicate that those defects have been fixed and vice versa. Team members do not need to be aware of where the defect was last processed. The latest data is available at all times, as long as the databases are synchronized frequently.

**StarTeam Microsoft Project Integration** Available with all licenses.

The interaction of the StarTeam Microsoft Project Integration and Microsoft Project make the jobs of both project planners and team members easier. Project planners use Microsoft Project to list the tasks that workers must perform. After exporting the tasks to StarTeam, they can gather information about the work accomplished by each team member in StarTeam, rather than communicating individually with each team member.

**File Compare/Merge** File Compare/Merge is a graphical compare/merge tool delivered with the StarTeam Cross-Platform Client. It enables you to compare a file dynamically with the file in the repository, and manually or automatically merge the content of the two files. File differences are highlighted in the File Compare/Merge panes using a configurable color scheme, and action buttons display in the highlighted areas to simplify the merging process.

**View Compare/Merge** View Compare/Merge is a comprehensive tool for comparing and merging views available with the StarTeam Cross-Platform Client. There are two versions of View Compare/Merge:

**Graphical** Provides interactive comparison and merging with per-item and per-folder interaction, allowing you to carefully control which items are compared and how each difference is resolved

**Command-line** Enables batch/shell-directed sessions.

# Products Included with StarTeam Enterprise Advantage Licenses

In addition to the products included with StarTeam Enterprise licenses, StarTeam Enterprise Advantage licenses also include the products listed below. The installation instructions for some products are not in this consolidated installation guide, but are located in the respective guide of that product.

<b>MPX Cache Agent</b>	A root MPX Cache Agent monitors the server's repository for file content and object properties. Via Message Broker, the data is pushed to remote MPX Cache Agents that are network-near to members of dispersed teams, improving the speed with which users access the data they need.
<b>StarTeam Extensions</b>	<p>StarTeam Extensions enables clients to take advantage of workflow and custom toolbar applications. The StarTeam Extensions files must be checked into the StarFlow Extensions project on each server configuration. If there is no StarFlow Extensions project, you need to create one.</p> <p>StarTeam Extensions also provides API documentation and samples.</p>
<b>StarTeam Workflow Designer</b>	Use the StarTeam Workflow Designer to create workflows for specific artifact types (such as change requests/defects) per project or even per view.
<b>StarTeam Notification Agent</b>	The StarTeam Notification Agent runs on the same computer as the StarTeam Server (or on a network-near computer) so that it can monitor the server and send notifications set up in your workflow.
<b>Search</b>	Search allows users to perform ad hoc queries across servers and projects. The query results reflect the access rights of the user logged on to Search so information is shared across the organization without compromising security.
<b>Datamart*</b>	<p>StarTeam Datamart is a complementary product to the StarTeam Server. StarTeam Datamart uses the StarTeam SDK to communicate with the StarTeam Server to create a reporting database that you can use with popular third party reporting applications such as Crystal Reports and Business Objects (reporting applications are not included with StarTeam Datamart). StarTeam Datamart extracts data from a StarTeam Server and places the data into a relational database, where reporting tools can access it. StarTeam Datamart can extract information from every project, every view in each project, every folder in each view, and every item in each folder, and labels, links, and history for each item. You can restrict extraction of data to a particular project and view or only extract certain tables.</p> <p>Datamart stores the data in any StarTeam supported database..</p> <p>The product comes with both an Extractor (for an initial retrieval) and with a Synchronizer to update existing data sets.</p>
<b>TeamInspector*</b>	TeamInspector is a continuous integration build server and build inspection tool. It works with StarTeam, Subversion, Perforce, and ClearCase. It requires the use of a database: Microsoft SQL Server 2005 SP3, Oracle 10g Release 2 version 10.2.0.4, or Apache Derby 10.4.2.0 or later.
<b>Rhythm*</b>	<p>Rhythm is an Agile project tracking tool designed to allow you to:</p> <ul style="list-style-type: none"><li>• Organize, prioritize, and manage your Agile teams' backlogs.</li><li>• Plan your sprints, task out the work, and then track progress throughout the sprint.</li><li>• Get comprehensive visibility of all your Agile assets.</li></ul>

\* Can be purchased separately and added to the Enterprise package.

## About Source Control

### Source Control Basics

Each source control system consists of one or more centralized repositories and a number of clients. A repository is a database that contains not only the actual data files, but also the structure of each project you define.

Most source control systems adhere to a concept of a logical project, within which files are stored, usually in one or more tree directory structures. A source control system project might contain one or many IDE-based projects in addition to other documents and artifacts. The system also enforces its own user authentication or, very often, takes advantage of the authentication provided by the underlying operating system. Doing so allows the source control system to maintain an audit trail or snapshot of updates to each file. By storing only the differences, the source control system can keep track of all changes with minimal storage requirements. When you want to see a complete copy of your file, the system performs a merge of the differences and presents you with a unified view. At the physical level, these differences are kept in separate files until you are ready to permanently merge your updates, at which time you can perform a commit action.

This approach allows you and other team members to work in parallel, simultaneously writing code for multiple shared projects, without the danger of an individual team member's code changes overwriting another's. Source control systems, in their most basic form, protect you from code conflicts and loss of early sources. Most source control systems give you the tools to manage code files with check-in and check-out capabilities, conflict reconciliation, and reporting capabilities. Most systems do not include logic conflict reconciliation or build management capabilities.

Commonly, source control systems only allow you to compare and merge revisions for text-based files, such as source code files, HTML documents, and XML documents. StarTeam stores binary files, such as images or compiled code, in the projects you place under control. You cannot, however, compare or merge revisions of binary files. If you need to do more than store and retrieve specific revisions of these types of files, you might consider creating a manual system to keep track of the changes made to such files.

### Repository Basics

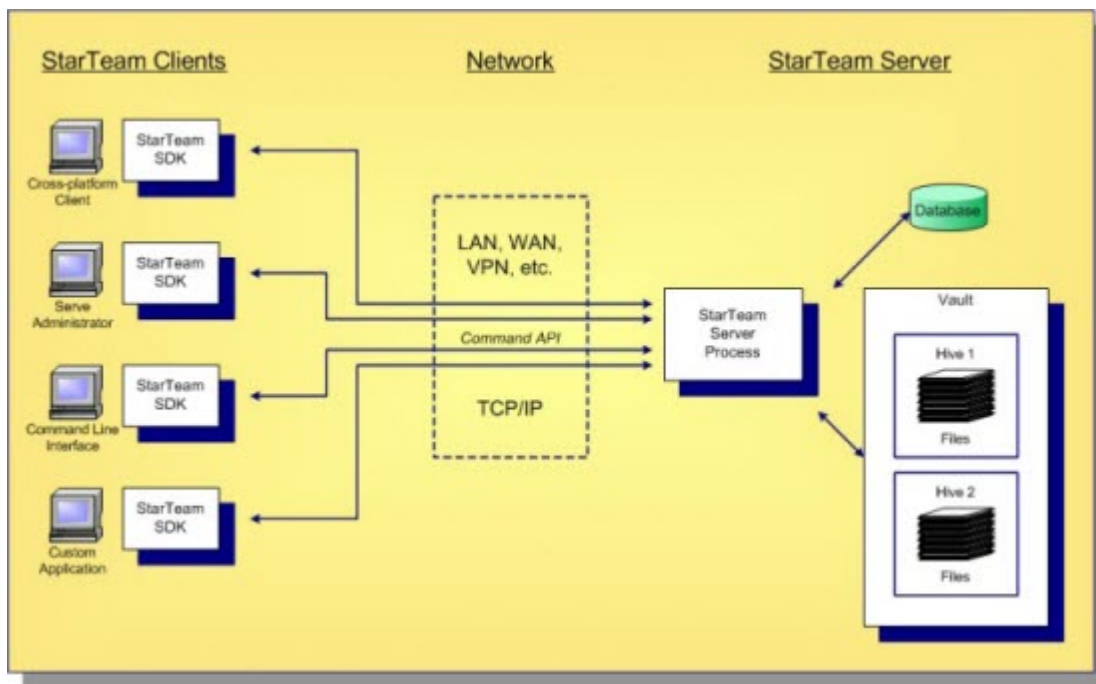
Source control systems store copies of source files and difference files in some form of database repository. In some systems, such as CVS or VSS, the repository is a logical structure that consists of a set of flat files and control files. In other systems, such as StarTeam, the repositories are instances of a particular database management system (DBMS).

Repositories are typically stored on a remote server, which allows multiple users to connect, check files in and out, and perform other management tasks simultaneously.

With StarTeam, you create a server configuration to identify a repository for StarTeam projects. Each server configuration acquires its own set of projects as they are created. The Server can run any number of server configurations. Because each server configuration must use a database, you need to make sure that you establish connectivity not only with the server, but also with the database instance.

## Standard StarTeam Architecture Overview

The standard architecture represents the minimal components present in a StarTeam instance: a StarTeam Server process managing a vault and a database and one or more StarTeam clients. With just these components, all basic StarTeam functionality is available. The core components of the standard StarTeam architecture are depicted below.



StarTeam employs a client/server architecture. The StarTeam Cross-Platform Client, **Server Administration** tool, and StarTeam Command Line Tools are examples of bundled StarTeam clients. StarTeam clients use the freely available StarTeam SDK, so you can write custom applications that have access to the same features as the bundled clients. The SDK is fully featured in Java, .NET, and COM, allowing you to write custom applications for any environment. A single StarTeam client can have multiple sessions to any number of StarTeam Servers.

All StarTeam clients connect to a StarTeam Server process using TCP/IP, so virtually any kind of network can be used: LAN, WAN, VPN, or the public Internet. StarTeam uses a proprietary protocol called the *command API*, which supports compression and multiple levels of encryption. The command API has been optimized to support high performance, automatic reconnect, delta check-out for slow connections, and other important features.

A single deployment instance of StarTeam is known as a *server configuration*, usually shortened to just *configuration*. The persistent data of a configuration consists of a database and a *vault* and is managed by a single StarTeam Server process. The database holds all metadata and non-file artifacts, whereas file contents are stored in the vault. The database can be any of the supported databases and it can reside on the same machine as the StarTeam Server process or a separate machine. The StarTeam database and vault can be backed-up dynamically, while the server is in use. This supports 24 x 7 operations that want to minimize down time.

StarTeam's vault is a critical component that affects performance and scalability. In contrast to the traditional delta storage technique, StarTeam's vault uses an innovative architecture designed for scalability, performance, high availability, and dynamic expandability. Today, customers are storing up to a terabyte of data in a single StarTeam vault, but it was designed to store content up to a petabyte and beyond.

Within the vault, files are stored in containers known as hives. A hive is a folder tree containing archive and cache files on a single disk volume. Hives can be dynamically added on existing or new disk volumes, thereby allowing virtually unlimited capacity. StarTeam stores each file revision in a separate archive file in a manner that minimizes space usage as well as duplicate content. StarTeam's vault uses less space than delta-based storage. In certain cases where it is more economical to send file deltas to clients instead of full versions, StarTeam generates and caches delta files. However, in most cases sending full versions is more economical.

## Contacting Support

Micro Focus is committed to providing world-class services in the areas of consulting and technical support. Qualified technical support engineers are prepared to handle your support needs on a case-by-case basis or in an ongoing partnership. Micro Focus provides worldwide support, delivering timely, reliable service to ensure every customer's business success.

For more information about support services, visit the Micro Focus SupportLine web site at <http://supportline.microfocus.com> where registered users can find product upgrades as well as previous versions of a product. Additionally, users can find the Knowledge Base, Product Documentation, Community Forums, and support resources.

When contacting support, be prepared to provide complete information about your environment, the product version, and a detailed description of the problem, including steps to reproduce the problem.

For support on third-party tools or documentation, contact the vendor of the tool.

## StarTeam Eclipse Plugin

StarTeam provides an automated and comprehensive software configuration management (SCM) system for supporting the management of assets and application life cycle tasks, all from within a single repository.



**Note:** If your Internet access is limited by network security, or if your computer is protected by a personal firewall, the Web-based links in this Help system might not function properly.

StarTeam facilitates both local and remote team collaboration through the use of tightly integrated components commonly used during product development. It allows you to:

- Store everything related to a group of projects in the same place. For example, you can store test and binary files, change requests and discussions about the project.
- Access information and collaborate via a LAN, WAN or the Internet securely.
- Improve project teamwork.
- Make testing easier and save maintenance time because of the tight integration between change requests and code and content files.
- Makes tracking requirements easier because of the tight integration between requirements and code and content files.
- Makes project management easier because the tight integration between tasks and code and content files.
- Keep track of who did what and when.
- Take advantage of true client/server access to data.

## Getting Started

Includes basic concepts about software change management and information on features and functionality to get you started with using the StarTeam Eclipse Plugin.

## Logging On

Before you can use StarTeam from within Eclipse, you must create a connection to a StarTeam server configuration and then log on to that server configuration.

StarTeam uses a standard Eclipse feature to store password information. The Eclipse encryption tool keeps password information securely hidden.

1. Open the Server Explorer.

2. Right click on the server configuration that you wish to access, and choose **Log On** from the context menu. The **Log On** dialog box opens.
3. Type a **User name** and **Password** in the appropriate text boxes.  
Passwords are case sensitive and may have length restrictions.
4. Check **Save as default credentials for this server**.
5. Click **OK**.

Once you are connected to a StarTeam server configuration, the name of the logged on user displays in the Server Explorer and in the status bar at the bottom of the Eclipse UI.



**Note:** The **Log On** dialog box does not use cached logon data. If you need to log on as a different user, run the **Log On** command again from the context menu.

## Logging Off

1. Open the Server Explorer.
2. Right click on the server configuration that you wish to access, and choose **Log Off** from the context menu. A confirmation dialog box displays asking if you want to completely log off this server.
3. Click **Yes** to continue.

StarTeam closes all connections to the specified server configuration that you have logged into during this session.

## Tour of the UI

This section describes the Perspectives and Views that make up the StarTeam Eclipse Plugin user interface.

### Perspectives

In the StarTeam Eclipse Plugin, you can access the StarTeam Activity and StarTeam Classic perspectives by performing one of the following:

- Selecting **Window > Open Perspective** and choosing the desired StarTeam perspective from the menu.
- Selecting **Window > Perspective > Open Perspective > Other** and choosing the desired StarTeam perspective from the StarTeam tree node.

#### StarTeam Activity Perspective

The **StarTeam Activity** perspective provides a suggested UI layout for activity-driven development. You can use the views in this perspective when working with and editing your files and using StarTeam as your source control provider.

The perspective shows the item and details views side-by-side and provides editor space so you can compare file edits and review changes before synchronizing with the StarTeam repository. It also provides an activity-driven development approach enabling you to select assigned action items, work on them in the editor area, change the status of the process item, and/or check in the work related to it.

The **StarTeam Activity** perspective opens all views provided in the **StarTeam Classic** perspective including the Editor view.

By default, this perspective includes the following views:

- **Editor**
- **Server Explorer**
- **Package Explorer**

- **Navigator**
- **Change Request**
- **Requirement**
- **Task**
- **Topic**
- **Audit Log**
- **Properties**
- **History**
- **Label**
- **Link**
- **Reference**

When opening any of the perspectives or views, a main menu command displays along with additional StarTeam-specific toolbar buttons.



**Note:** Once connected to a StarTeam server configuration, the Navigator and Package Explorers display the StarTeam project or view name in brackets next to the root folder name. For example, if your root Eclipse project folder is *MyProject*, your project name is *StarDraw*, and your view name is *MyView*, the Navigator and Package Explorers display: `MyProject [StarDraw/MyView]`

### StarTeam Classic Perspective

The **StarTeam Classic** perspective is provided to mimic the Cross-Platform Client and make users accustomed to the client feel at home. Choose this perspective if you want to work as if you are working in the client. You can use this perspective for reviewing Change Requests, Topics, or other components that require a large viewing space.

The **StarTeam Classic** perspective opens all views provided in the **StarTeam Activity** perspective, excluding the Editor view.

By default, the **StarTeam Classic** perspective provides the following views:

- **Server Explorer**
- **Package Explorer**
- **Navigator**
- **Change Request**
- **Requirement**
- **Task**
- **Topic**
- **Audit Log**
- **Properties**
- **History**
- **Label**
- **Link**
- **Reference**

When opening any of the perspectives or views, a main menu command displays along with additional StarTeam-specific toolbar buttons.



**Note:** Once connected to a StarTeam server configuration, the Navigator and Package Explorers display the StarTeam project or view name in brackets next to the root folder name. For example, if your root Eclipse project folder is *MyProject*, your project name is *StarDraw*, and your view name is *MyView*, the Navigator and Package Explorers display: `MyProject [StarDraw/MyView]`



## Views



**Important:** The term *view* is used here in terms of the Eclipse UI. This term should not be confused with the StarTeam usage of *view*. In StarTeam, a *view* consists of an application folder tree view and the items associated with each folder in that tree.

You can access the StarTeam views by performing one of the following:

- Selecting **Window > Show View** and selecting the desired view from the menu.
- Selecting **Window > Show View > Other** and selecting the desired view from the StarTeam tree node.

When opening any of the perspectives or views, a main menu command displays along with additional StarTeam-specific toolbar buttons.

When you create a project, an initial or root view of that project is also created. This initial view, which has the same name as the project, consists of the root folder and a tree view of the child folder. It is always read/write.

The root view is called a dynamic view, because it shows all items in the project as they change, making it ideal for collaborative development.

To accommodate both user and project needs, however, the application enables you to create additional views of a project based on this view. These additional views may contain some or all of the contents of the original view and may behave differently.

For example, you might use views to:

- Implement the same folder tree for both the 2.5 release and the 3.0 release of a product. The easiest way to do this is by creating a new view for the 3.0 release based on the view used for the 2.5 release.
- Limit the portion of the project that certain team members see. Developers might need to see only the project's source code folder and its child folders; marketing personnel might need to see only the marketing folder and its child folders; and so on. Each of these views can have a different folder as its root.
- Support branching and parallel development. By branching files and other data in a new view, you organization can start to work on the 2.0 version of a product without hampering the creation of service packs for the 1.0 version.

Views represent configurations of items and support different development baselines of the same code base. If desired, views can be compared and merged. For example, you might want eventually to merge files from 4.5 Maintenance and 5.0 New Development into the Baseline view. Views are highly flexible. For example, they can be re-configured to show items as they existed at an earlier point in time or based on a view label or associated promotion state. Rollback views are read-only, as they show a precise state of the items, and no longer permit changes.

### Server Explorer

The top-level node is the unique name that you provide to the server configuration in the **Share Project** wizard. Expanding the root server configuration node reveals StarTeam projects, views, folders, and files. You can open the Server Explorer by doing one of the following:

- Select **Window > Show View > Server Explorer**
- Select **Window > Show View > StarTeam Other**, and choosing **Server Explorer** from the tree node.

A connection to the server configuration is present if you see a logged in name next to the server configuration name in the Server Explorer or in the status bar at the bottom of the workbench.

The Server Explorer view displays the server configuration, project, view, and folder tree view. With this view you can perform the following operations:

- **Display information using icons:** A specific icon precedes each server, view, project and folder name. Selecting one of the nodes in this tree view causes all other StarTeam-related views to update with information relevant to this selection.

- **Expand and collapse branches:** You can expand or collapse branches in the Server Explorer. A plus sign identifies a collapsed branch. Clicking it expands the branch. A minus sign identifies an expanded branch. Clicking it collapses the branch.
- **Right-click for context menu commands:** After selecting a node in the Server Explorer, right-click to display a context menu offering commands specific to the selected node.

### Server Explorer Context Menu Actions

At the server configuration level, context menus enable you to:

- Create new StarTeam projects, views, and folders
- Modify existing server configuration connection properties
- Delete a connection to a server configuration
- Log on to the server configuration

At the project level, context menus enable you to:

- Open projects in various StarTeam views, including Change Request, Requirement, Task, and Topic views
- Create new views and folders
- Delete projects
- Create desktop shortcuts (.stx) to quickly launch the Cross-Platform Client
- Modify project properties including options like keyword expansion and process rules

At the view level, context menus enable you to:

- Open views in various StarTeam views, including Change Request, Requirement, Task, and Topic views
- Create new views and folders
- Create, delete, and modify labels and promotion states
- Modify view properties, including the view name, branching settings, and the working folder configuration
- Delete views
- Create desktop shortcuts (.stx) to quickly launch the Cross-Platform Client

At the folder level, context menus enable you to:

- Create new folders, change requests, requirements, tasks, or topics
- Open folders in various StarTeam views, including Change Request, Requirement, Task, and Topic views
- Open the folder locally
- Check out a folder as a workspace project, to include in an existing workspace project, or configure with the **New Project** wizard
- Create URLs and HTML representations and copy them to the Windows Clipboard and paste them into email applications, Word documents, Excel spreadsheets, and so on to quickly open the folder in StarTeam
- Create, attach, detach, and modify properties for labels
- Modify folder properties including the folder name and files to exclude from folders
- Add, edit, enable, or disable custom fields
- Modify folder configuration
- Delete folders
- Create, complete, and cancel links
- Create desktop shortcuts (.stx) to quickly launch the Cross-Platform Client

At the file level, context menus enable you to:

- Open files in various StarTeam views, including Change Request, Requirement, Task, and Topic views
- Open the local folder containing the file

- Send file properties information as an email message, along with additional text.
- Set the lock status
- Create URLs and HTML representations and copy them to the Windows Clipboard and paste them into email applications, Word documents, Excel spreadsheets, and so on to quickly open the file in StarTeam
- Create, attach, and detach labels
- Flag and remove flags from files
- Delete files
- Create, complete, and cancel links
- Create desktop shortcuts (.stx) to quickly launch the Cross-Platform Client
- View and modify file properties, such as the file name, description, and file storage options

### Annotate View

Annotations show historical information about changes that were made to a text document. The **Annotate** view is a tabbed pane which is displayed in the same location as other StarTeam views, such as the **Detail**, **History**, **Links**, and **Labels** views. It displays the contents of the selected text file, a working folder hyperlink to the file in the Server Explorer, and annotation information in the left margin which contains the following information:

- The name of each user who wrote or changed specific lines, and which lines were changed.
- The dot notation of the file revision in which each change was made.
- A hyperlink to the properties for the associated process item, if one was linked to the file revision.
- An Info icon which, when clicked, displays the details about the file revision and the associated process item.

You can open the **Annotate** view by taking the following actions:

- Select **Window > Show View > Annotate**
- Select **Window > Show View > Other**, and choose **Annotate** from the StarTeam tree node.

### Annotate View Toolbar and Menu Actions

The following actions are available from the Annotate View toolbar and list box:

- Link (unlink) the view to the selected StarTeam item
- Refresh
- Back to previous contents
- Forward to next contents
- Open StarTeam Preferences

### Editor Annotations

The StarTeam annotate feature also provides a **Team > Show Annotation** context-menu action which displays annotation information in the left margin of the editor, and can be invoked for any shared file or repository file. This menu action is available by right-clicking a text file in the **Server Explorer**, **Navigator**, or **Package Explorer**. Annotations utilize the StarTeam quick diff provider for displaying revision information, and StarTeam requests that you switch to it for the current editor.

Annotations are displayed in the left panel of the editor, and are synchronized with the **Annotate** view so that when you select an annotation in the editor, it selects that annotation in the **Annotate** view. The default coloring for the left gutter is by revision age. The color gradually brightens from dark (latest changes) to light (earliest changes). You can modify the coloring to distinguish authors who contributed the change, or display both revision age and authors.

The contents of the generic **History** view are also synchronized with the annotations in the editor. Selecting a revision in the **History** view highlights the left gutter annotation borders of all the lines in the document that were contributed by the selected revision, and the right margin shows the range of the

changes created by the selected revision. Conversely, selecting an annotation in the left gutter of the editor highlights the corresponding revision in the **History** view.

The fields in the left gutter are sensitive to mouse moves and show hover information with more details about the contributed changes. This information is the same information obtained by clicking the **Information** icon in the **Annotate** view. Pressing **F2** over a hover window gives it focus, allowing you to read the details or use the hyperlinks to possible process items linked to the revision. Clicking a link to a process item in the annotation information also selects and displays that item in the **Change Request**, **Task**, or **Requirement** view.

### Annotation Context-Menu Actions in the Editor

There are no context-menu actions in the **Annotate** view, however, there are several actions available from the **Annotation** pane of the editor using the following Revisions context-menu items:

- Hide or display revision information.
- Display the border color by the revision date.
- Display the border color by the author who made the changes.
- Combine the border coloring to show the revision date and the author.
- Show the author's name.
- Show the revision ID (dot notation revision number).

### Audit Log View

The **Audit Log** view provides a chronological record accumulating data about the actions performed on folders, files, requirements, change requests, tasks, and topics.

You can open the **Audit Log** view by doing one of the following:

- Selecting **Window > Show View > Audit**
- Selecting **Window > Show View > Other**, and choosing **Audit** from the StarTeam tree node.

When you open the **Audit Log** view, a list of audit logs displays for the selected view.

All audits shown in the **Audit Log** view:

- Are attached to the folder selected from the Server Explorer.
- Match the filter selected from the **Filter** list box.
- Match the depth specified by the **All Descendants** button.

### Using the Audit Log List

Audit log entries cannot be moved, shared, modified, or branched in StarTeam. However, a few operations can be performed on these items. For example, you can search for items in the **Audit Log** view by clicking on the list box arrow of the Audit Log view toolbar, and selecting **Find**, **Find Next**, or **Find Previous** to locate items in the Audit log list with a specified value in a column.

### Filter and Sort Audit Log Data

You can sort the audit log by the data in a column and change the display based on a predefined filter available in the Audit Log view toolbar.

You can click on an audit column header to perform a sort based on the value in that column. The sort is usually in ascending numeric or alphanumeric order, depending on the data. To change the sort order from ascending to descending (or vice versa), click the header a second time. A triangle indicating the direction of the sort appears on the header of the primary sort column.

### Default Audit Log View Columns

You can view the following default columns in the **Audit Log** view:

- **User**– Person performing the action.

- **Created Time** – Date and time of the action.
- **Class Name** – Type of object, such as file, folder, change request, or topic
- **Event** – Name of the action, such as Created, Added, Deleted, Modified, Branched, Moved From, Shared, Locked, Unlocked, Moved to, or Lock broken.
- **Item** – Item number associated with the Class Name. That is, the type of item created, such as a task, topic, change request, and so on.
- **View** – Name of the view in which the object is located.
- **Project** – Name of project where the object is located.

### Audit Log View Toolbar and Context Menu Actions

The Audit Log view toolbar and context menu allow you to:

- Filter log entries with predefined filters
- Search (CTRL+F) audit log entries
- Link (unlink) the view to the selected StarTeam item
- Optionally show all descendants to determine the depth for which the client displays information
- Create URLs and HTML representations and copy them to the Windows Clipboard and paste them into email applications, Word documents, Excel spreadsheets, and so on to quickly open to the item of interest in StarTeam
- Create desktop shortcuts (.stx) to quickly launch the Cross-Platform Client

### Change View

The **Change** view shows all the actions performed on a process item or workspace (check-in) change package.

You can open the **Change** view by taking one of the following actions:

- Select **Window > Show View > Change**
- Select **Window > Show View > Other**, and choose **Change** from the Server Explorer tree node.

Every time you make an update to a file or folder, there will be an entry in the details of the new **Change** view. It shows a description of the change or action performed, for example "Workspace Changes on <date>". The **Change** view also lists the process item used for the check in. Whenever a process item is used or specified for check in, a change package is created even if the process item is not directly changed, for example, "set to fixed as part of the check in".

### Change View Context Menu Actions

The Change view context menu enables you to complete the following tasks:

- View historical information about changes made to a file.
- Highlight a selected item in the Server Explorer view. This option cannot be used on custom components.
- View properties of two items to compare changes between them.
- View contents of two items to compare changes between them.
- View properties of the workspace (check-in) change package.

### Change Request View

The Change Request view provides a powerful search and/or reporting mechanism that allows change requests and suggestions about possible improvements to a project to be entered, assigned to a team member, and tracked.

You can open the **Change Request** view by taking one of the following actions:

- Select **Window > Show View > Change Request**

- Select **Window > Show View > Other**, and choose **Change Request** from the StarTeam tree node.

When you open the **Change Request** view, a list of change requests displays.

All change requests shown in the **Change Request** view have the following characteristics:

- Are attached to the folder selected from the Server Explorer.
- Match the filter selected from the **Filter** list box.
- Match the depth specified by the **All Descendants** button.

### Using the Change Request List

From the Change Request list that displays in the **Change Request** view, you can perform a variety of operations, including the following options:

- **View unread items:** To see the change requests for which you are responsible but have not reviewed, look for the change requests in boldface type. The change requests in regular type are those that you have read or those for which you are not responsible.
- **View properties:** To open the **Properties** dialog box for a specific change request, double click on it.

### Filter and Sort Change Request Data

You can sort the change requests by the data in a column and change the display based on a predefined filter available in the Change Request view toolbar.

You can sort change requests by clicking on a column header to sort the displayed change requests based on the value in that column. The sort is usually in ascending numeric or alphanumeric order, depending on the data. The ascending order for Severity displays from Low to Medium to High. The ascending order for Status reflects the life cycle of the change request, so these items display from New to Closed. The ascending order for Priority displays from No to Yes. To change the sort order from ascending to descending, click the header a second time. A triangle indicating the direction of the sort appears on the header of the primary sort column.

### Default Change Request View Columns

You can view the following default columns in the **Change Request** view:

- **CR Number:** Number that StarTeam assigned to the change request when it was first submitted.
- **Synopsis:** Short explanation of the change request.
- **Type:** Defect or Suggestion.
- **Status:** Indication of the change request life cycle. Status classifications include New, Open, Cannot Reproduce, As Designed, Fixed, Documented, Is Duplicate, or Deferred.
- **Severity:** Seriousness of the change request. Severity can be Minor, Normal, Major, or Critical.
- **Responsibility:** Person who is responsible for the change request.
- **Addressed In:** Build in which the change request is resolved.
- **Addressed By:** Name of user who resolved the change request. A change request has been resolved when its status becomes Cannot Reproduce, As Designed, Fixed, Documented, or Is Duplicate.
- **Last Build Tested:** Build in which the change request occurs.
- **Work Around:** Solution to a change request other than a fix.
- **Modified Time:** Time the change request was last modified.
- **Priority:** Indication that the change request is or is not a Priority.
- **Description:** Complete explanation of the defect or suggestion, including the steps to be taken to reproduce the problem.
- **Test Command:** Command that can be used to test the change request's solution.
- **Fix:** Solution to the problem addressed by the change request.
- **Entered On:** Date and time the change request was submitted.
- **Entered By:** Person who submitted the change request.

## Change Request View Toolbar and Context Menu Actions

In addition to the other capabilities described in this section, the Change Request view toolbar and context menu enable you to use the following options:

- Apply predefined filters for viewing lists of change requests.
- Search (CTRL+F) for change requests.
- Link (unlink) the view to the selected StarTeam item.
- Optionally show all descendants to determine the depth for which the client displays information
- Select all items.
- Select items using a label or pre-defined query.
- Determine which fields to display in the view.
- Perform up to fourth order sorts.
- Create, edit, copy, or delete queries
- Save or reset current view filters
- Create, edit, save, or delete filters
- Create new change requests
- Open change requests in various views, including History, Label, Link, and Reference views
- Modify and review change request properties
- Email a text representation of the change request's properties, along with additional text. The information sent includes the fields displayed in the view.
- Compare the properties of two change requests.
- Set the lock status.
- Mark the item as read or unread.
- Add, edit, enable, or disable custom fields.
- Create URLs and HTML representations and copy them to the Windows Clipboard and paste them into email applications, Word documents, Excel spreadsheets, and so on to quickly open the item in StarTeam
- Create, attach, and detach labels.
- Flag and remove flags from files.
- Delete change requests.
- Create, complete, and cancel links
- Create desktop shortcuts (.stx) to quickly launch the Cross-Platform Client
- Set or clear the item as an active process item.

## Detail View

The **Detail** view displays a customized summary and text details of the properties for the selected item.

You can open the **Detail** view by taking either of the following actions:

- Selecting **Window > Show View > Detail**
- Selecting **Window > Show View > Other**, and choosing **Detail** from the StarTeam tree node.

## Detail View Toolbar and Menu Actions

The following actions are available from the **Detail** View toolbar and drop-down menu:

- Link (unlink) the view to the selected StarTeam item
- Refresh
- Back to previous contents
- Forward to next contents
- Open StarTeam Preferences

## Detail View Context-Menu Actions

No context-menu actions are available in the **Detail** view.

## History View

The History view displays historical information about the selected item. This view plays an important role in version control. After you select the appropriate file or item, the History view automatically displays a list of all the past revisions of that file or item. The revisions are listed from the tip revision to the initial revision. Double-clicking a revision displays its properties. Right-clicking displays a context menu that enables you to review the properties for the revision, to compare revisions, and to change the comment associated with that revision. For files, you can also view the file in the default editor or open in an appropriate application.

You can open the History view by taking one of the following actions:

- Select **Window > Show View > History**
- Select **Window > Show View > Other**, and choose **History** from the Server Explorer tree node.

Default columns provided in the History view include the following columns:

- **View** – View to which the file belongs.
- **Revision** – Number StarTeam assigned to the revision when it was checked in.
- **Author** – Person who checked in the revision.
- **Time** – Date and time of the revision.
- **Comment** – Reason for creating the revision.
- **Branch Revision** – Indicates the branch number for the item.

## History View Context Menu Actions

The History view context menu enables you to complete the following tasks:

- View content of revision in a default editor (files only)
- Display the properties of a revision
- Compare revisions
- Compare properties of revisions
- Change comments associated with revisions
- Create URLs and HTML representations and copy them to the Windows Clipboard and paste them into email applications, Word documents, Excel spreadsheets, and so on to quickly open the item in StarTeam
- Create desktop shortcuts (.stx) to quickly launch the Cross-Platform Client
- Open the local folder containing the selected file

## Label View

The Label view provides a list of labels assigned to a corresponding folder or item. In version control, the term *label* corresponds to the act of attaching a view or revision label (name) to one or more folders and/or items. You can attach a label to almost any type of StarTeam item.

You can open the **Label** view by taking one of the following actions:

- Selecting **Window > Show View > Label**
- Selecting **Window > Show View > Other**, and choosing **Label** from the StarTeam tree node.

StarTeam offers two types of labels:

- View labels, which are automatically and immediately attached to all folders and items in a view at the time they are created. They have multiple purposes, but are primarily used as a time stamp for the view and as build labels.



- Revision labels, which are not automatically attached to any item in the view. Instead, they are used to designate a set of folders or items within a view. For example, you might want to label a group of files that should be checked in and out together.

### **Label View Context Menu Actions**

The Label view toolbar and context menu allow you to:

- Link (unlink) the view to the selected item
- Attach/detach labels
- Modify label properties

### **Link View**

The **Link** view displays a connection between two folders, two items, or a folder and an item.

You can open the **Link** view by doing one of the following:

- Selecting **Window > Show View > Link**
- Selecting **Window > Show View > Other**, and choosing **Link** from the StarTeam tree node.

### **Link View Context Menu Actions**

The Link view toolbar and context menu enable you to accomplish the following tasks:

- Link (unlink) the view to the selected item
- Select linked items
- Attach/detach labels to/from linked files
- Float (allow the link source or target to change from tip revision to tip revision as new revisions are created) and pin (lock the link to the tip revision) links
- Delete links
- Modify link and source/target item properties

### **Reference View**

The **Reference** view indicates the relationships between an original folder or item and those that are based on it. By looking at the Reference view, you can decide whether the changes made to a folder or item in one location need to be applied elsewhere.

You can open the **Reference** view by doing one of the following:

- Selecting **Window > Show View > Reference**
- Selecting **Window > Show View > Other**, and choosing **Reference** from the StarTeam tree node.

### **Reference View Context Menu Actions**

The Reference view toolbar and context menu allow you to accomplish the following tasks:

- Link (unlink) the view to the selected StarTeam item.
- Modify the referenced item's properties.

### **Requirement View**

You can open the **Requirement** view by doing one of the following:

- Select **Window > Show View > Requirement**
- Select **Window > Show View > Other**, and choose **Requirement** from the StarTeam tree node.

When you open the **Requirement** view, a list of requirements displays.

All requirements shown in the **Requirement** view have the following characteristics:

- Are attached to the folder selected from the Server Explorer.
- Match the filter selected from the **Filter** list box.
- Match the depth specified by the **All Descendants** button.

### Requirement View Toolbar and Context Menu Actions

The Requirement view toolbar and context menu allow you to accomplish the following tasks:

- Apply predefined filters for viewing lists of requirements
- Search (CTRL+F) for requirements
- Link (unlink) the view to the selected item
- Optionally show all descendants to determine the depth for which the client displays information
- View in list or tree format
- Select all items
- Select items using a label or pre-defined query
- Determine which fields to display in the view
- Perform up to fourth order sorts
- Create, edit, copy, or delete queries
- Save or reset current view filters
- Create, edit, save, or delete filters
- Create new requirements and child requirements
- Open requirements in various views, including History, Label, Link, and Reference views
- Modify and review requirement properties
- Email a text representation of the requirement's properties, along with additional text. The information sent includes the fields displayed in the view.
- Compare the properties of two requirements
- Set the lock status
- Mark the item or thread as read or unread
- Add, edit, enable, or disable custom fields
- Create, attach, and detach labels
- Flag and remove flags from files
- Delete requirements
- Create, complete, and cancel links
- Set or clear the item as an active process item

### Task View

In the **Task** view, team members can easily see who should do what tasks and when tasks should be done. It also enables team members to see current task status, estimate the time required for a task, record the time spent, and compare estimated to actual times.

You can open the **Task** view by taking one of the following actions:

- Select **Window > Show View > Task**
- Select **Window > Show View > Other**, and choose **Task** from the StarTeam tree node.

When you open the **Task** view, existing tasks display. Double-clicking a task or subtask opens the **Properties** dialog box for the selected task.

All tasks displayed in the **Task** view have the following characteristics:

- Are attached to the folder selected from the Server Explorer.
- Match the filter selected from the **Filter** list box.
- Match the depth specified by the **All Descendants** button.

## Task View Toolbar and Context Menu Actions

The Task view toolbar and context menu allow you to complete the following tasks:

- Apply predefined filters for viewing lists of tasks
- Search (CTRL+F) for tasks
- Link (unlink) the view to the selected StarTeam item
- Optionally show all descendants to determine the depth for which the client displays information
- View in list or tree format
- Select all items
- Select items using a label or pre-defined query
- Determine which fields to display in the view
- Perform up to fourth order sorts
- Create, edit, copy, or delete queries
- Save or reset current view filters
- Create, edit, save, or delete filters
- Create new tasks and subtasks
- Add work records to tasks or subtasks
- Open tasks in various views, including History, Label, Link, and Reference views
- Modify and review task properties
- Email a text representation of the task's properties, along with additional text. The information sent includes the fields displayed in the view.
- Compare the properties of two tasks
- Set the lock status
- Mark the item or thread as read or unread
- Add, edit, enable, or disable custom fields
- Create URLs and HTML representations and copy them to the Windows Clipboard and paste them into email applications, Word documents, Excel spreadsheets, and so on to quickly open the item in StarTeam
- Create, attach, and detach labels
- Flag and remove flags from files
- Delete tasks
- Create, complete, and cancel links
- Create desktop shortcuts (.stx) to quickly launch the Cross-Platform Client
- Set or clear the item as an active process item

## Topic View

In the **Topic** view, team members can raise and discuss issues about a project, specific files, defects, and so on.

You can open the **Topic** view by taking one of the following actions:

- Select **Window > Show View > Topic**
- Select **Window > Show View > Other**, and choose **Topic** from the StarTeam tree node.

When you open the **Topic** view, existing topics and responses display. Double-clicking on a topic or response in the **Topic** view opens the **Properties** dialog box.

All topics and responses displayed in the **Topic** view have the following characteristics:

- Are attached to the folder selected from the Server Explorer or from one of the Eclipse Explorers.
- Match the filter selected from the **Filter** list box in the Topic view toolbar.
- Match the depth specified by the **All Descendants** option in the Topic view toolbar.

## Topic View Toolbar and Context Menu Actions

The Topic view toolbar and context menu allow you to:

- Apply predefined filters for viewing lists of topics
- Search (CTRL+F) for topics
- Link (unlink) the view to the selected item
- Optionally show all descendants to determine the depth for which the client displays information
- View in list or tree format
- Select all items
- Select items using a label or pre-defined query
- Determine which fields to display in the view
- Perform up to fourth order sorts
- Create, edit, copy, or delete queries
- Save or reset current view filters
- Create, edit, save, or delete filters
- Create new topics and respond to existing topics
- Open topics in various views, including **History**, **Label**, **Link**, and **Reference**.
- Modify and review topic properties
- Email a text representation of the topic's properties, along with additional text. The information sent includes the fields displayed in the view.
- Compare the properties of two topics
- Set the lock status
- Mark the item or thread as read or unread
- Add, edit, enable, or disable custom fields
- Create URLs and HTML representations and copy them to the Windows Clipboard and paste them into email applications, Word documents, Excel spreadsheets, and so on to quickly open the item in StarTeam
- Create, attach, and detach labels
- Flag and remove flags from files
- Delete topics
- Create, complete, and cancel links
- Create desktop shortcuts (.stx) to quickly launch the Cross-Platform Client
- Set or clear the item as an active process item

## Search Results View

Once you have entered the expression for which you would like to search you can specify the following:

- **Type** – Select the item type from the available list box. For example you can search just Topic items.
- **Search in** – Select the check box next to any server, view, project, or folder to be included in the search.
- **Look in** – Select how the item properties should narrow the scope of the search.
- **Options** – Select the appropriate check box to make the search recursive and the expression match text case.

The results of the StarTeam search display in the **Search Results** view.

## Item Editors

There are two ways to create and edit change requests, requirements, tasks, and topics:

- Dialog box property editors (modal)
- Embedded property editors (non-modal)

Each has its own advantages.

## Dialog Box Property Editors

Dialog box editors open in a separate window for creating new items, and for editing or checking the status of existing ones.

Dialog box property editors enable you to do the following:

- Create or modify an item without disturbing your workspace.
- Open a dialog box property editor from within another dialog box, such as the file **Check In** or **Check Out** dialog box.

For example, from the **Process Item** or the **Changes** area in the Check In dialog box, you might open the property dialog box for the process item you are using and copy its synopsis to the description field in the Check In dialog box. This is not allowed in embedded editor because the **Check In** dialog box itself is modal and blocks the workspace UI.

## Embedded Property Editors

Embedded property editors open in the Eclipse editor pane using the Eclipse forms look and feel. When an item is opened in an embedded editor, the embedded property editor takes precedence and the controls in the properties dialog box for that item are disabled.

Embedded property editors enable you to do the following:

- Continue to do other work in Eclipse while the embedded property editor is open.
- Open multiple embedded property editors at the same time.
- Drag text content from one property editor to another.
- Drag a block of text from a file in the editor to a field in an embedded property editor.
- Drag and drop files, such as images or log files, from the Server Explorer onto the attachment tab of an embedded property editor.
- Quickly open item property editors by dragging one or more items from the Item view into the editor pane.
- View warnings and error messages in the header of the active tab page.
- See when changes to an item are unsaved, represented by an asterisk next to the editor title on the tab.

## Fields

You can display both common and advanced fields as columns in the **Change Request**, **Requirement**, **Task**, and **Topic** views or use in queries along with relational operators that you can use with those fields to define conditions. The StarTeam Eclipse Plugin comes preconfigured to use the default fields that normally display in the Cross-Platform Client. You can sort fields by clicking on a column header in the view to sort the data in the selected column. You can display additional fields using **Show Fields** command found in the list box toolbar menu for the **Change Request**, **Requirement**, **Task**, and **Topic** views.

## Sharing Workspace Projects

Follow the high-level steps below when using the **Share Project** wizard.

These steps assume that:

- You have not yet created a connection with a StarTeam Server configuration.
- You have already created multiple Eclipse projects in the Workspace to share.
- You have pre-existing StarTeam projects created for the StarTeam Server configuration that you are sharing with the Eclipse projects.

1. Open the **Share Project** wizard.
2. If necessary, create a new StarTeam connection.
3. Log on to a StarTeam Server.
4. Enter repository location information.

5. Synchronize your workspace project with the StarTeam repository.
6. Check in the workspace project files to the repository.

## Create Projects and Synchronize Local Files

At a high-level, creating a new StarTeam project and associating an existing workspace project with it involves the following steps:

These steps assume that:

- You have not yet created a connection to a StarTeam server configuration.
- You have already created an Eclipse project in the Workspace to share.

## Checking Out Existing Folders

1. Open the StarTeam perspective.
2. Choose **Window > Perspective > Open Perspective > Other**.
3. Select StarTeam Classic or StarTeam Activity from the list, and click **OK**.
4. Open the Server Explorer, and navigate to the appropriate StarTeam project, view, and folder.

You can multi-select folders to check out in the Server Explorer.



**Note:** If you have not logged on to the server, right click on the server configuration node, and choose **Log On**. The status bar at the bottom of the Workbench reflects the logged on user name.

5. Right-click and choose **Check Out As Project**. The **Check Out As** wizard opens.
6. Choose one of the following options:

Option	Description
<b>Check out as project in the workspace</b>	Enter a project name in the text box provided or use the default folder name, and click <b>Next</b> . Select a location where you want to check out the files to, and click <b>Next</b> .
<b>Check out into existing project</b>	Click <b>Next</b> to choose a valid target project, and click <b>Finish</b> .
<b>Check out as a project configured using New Project Wizard</b>	Click <b>Finish</b> , and follow the steps in the <b>New Project</b> wizard.



**Note:** If you want to use a location within the workspace that is not the default (that is, you select either the second or third option), then you must use the StarTeam Cross-Platform Client to check out the project and then import it from within the StarTeam Eclipse Plugin.

7. If you chose to **Check out as project in the workspace**, specify any check out options as follows:
  - Expand **Reference By**, and select one of the following options for the files you wish to check out:
    - **Current revision**: The most current (tip) revision.
    - **Label**: A specific file revision. The existing view and revision labels are listed in reverse chronological order based on the time at which they were created. The view labels precede the revision labels in the list.
    - **Promotion state**: A specific promotion state.
    - **Configuration as of**: The revision that was the tip revision at the specified date and time. Click the **Date/Year** button to use the calendar, and specify the time by typing in the time or using the spin boxes.
  - Expand **Lock Status**, and select one of the following lock options:
    - **Unlocked**: Releases your lock on the files after check-in.
    - **Exclusive**: Indicates that you intend to make further changes to the files.

- **Non-exclusive:** Indicates that you are working on the files and may possibly make changes.
  - **Keep Current:** Retains the current lock status. Selected by default.
  - *Optional:* Expand **EOL Conversion**, and choose one of the following:
    - **None** or another button to change your current EOL conversion setting.
    - For Windows, the EOL marker is **CR-LF** (carriage return/line feed).
    - For UNIX, it is **LF** (line feed).
    - For Macintosh operating systems, it is **CR** (carriage return).
  - Expand **Advanced** to optionally select an appropriate **File Encoding** from the drop-down list to support keyword expansion for non-English code pages.
8. Click **Next**. The **Mapping preferences** page opens.
  9. Choose your desired settings, or accept the default settings, and click **Finish**.

## Setting Preferences

To review a description of the preferences you can set, see the links at the end of this topic.

1. Choose **Window > Preferences** from the main menu. The **Preferences** dialog box opens.
2. In the resulting dialog box, expand **Team > StarTeam**.
3. Select the node containing the options that you want to change.
4. Make the desired changes, then click **OK**.

## Opening Perspectives and Views

1. Choose **Window > Perspective > Open Perspective > Other** from the main menu. The **Open Perspective** dialog box opens.
2. Select either the StarTeam Activity or StarTeam Classic perspective.
3. Click **OK**.

The specified perspective opens.

## Configuring Capabilities

This task describes how to enable and disable StarTeam as your source control provider plug in. By default, when you install StarTeam, it is automatically enabled.

To enable and disable StarTeam capabilities

1. Choose **Window > Preferences** from the main menu. The **Preferences** dialog box opens.
2. In the resulting dialog box, expand **General** and select **Capabilities**.
3. Click **Advanced**. The **Advanced** dialog box opens.
4. Take one of the following actions to complete your task:
  - To disable StarTeam capabilities, expand **Team**, and clear the **StarTeam Support** check box.
  - To enable disabled StarTeam capabilities, expand **Team**, and check **StarTeam Support**.
5. Click **OK**. The **Advanced** dialog box closes.
6. Click **OK**. The **Preferences** dialog box closes.

## Creating a New Connection

1. Select **Create a new StarTeam** connection and click **Next**.
2. Type a unique, easy-to-remember description in the **Server description** text box.  
The server description is not case-sensitive and can contain colons ( : ).

3. Type the computer name or IP address in the **Server address** text box.  
See your administrator for the server address, protocol, and endpoint information.
4. Type the endpoint (TCP/IP port number) associated with the protocol in the **TCP/IP endpoint** text box.
5. Click **Test Connection**.
6. Choose to specify any of the following optional settings:
  - Check the **Save Password** option to enable the **User Name** and **Password** text boxes.
  - Select an **Encryption** type to encrypt data transferred between your workstation and the server. Encryption protects files and other project information from being read by unauthorized parties over unsecured network lines. The encryption types are ordered (top to bottom) based on speed. Each type is slower, but safer, than the type that precedes it.
  - Check the option to **Compress transferred** data if you want to use compression.
7. Click **Next** when you are finished.

## Removing Connections

1. Open an Eclipse Explorer.
2. Select the project node that you wish to remove StarTeam sharing with.
3. Right-click on the project node, and choose **Team > Remove Share**.

## Configuring Profiles

Use this task to configure profiles for an existing StarTeam server configuration.

1. Open the Server Explorer.
2. Right-click on the root node for the desired server configuration connection, and choose **Properties**. The **Properties** dialog box opens.
3. Expand **Server Description**, and choose **MPX Profiles**. The **Properties** dialog box lists each profile defined in the Event Transmitter XML file for this server configuration.
4. Select the desired profile.  
If you wish to restore the client default profile for this configuration, click **Restore Default**.
5. Click **Close** on this dialog, and then click **OK** on the **Properties** dialog box.

## Deleting Server Configurations

1. Open the Server Explorer.
2. Select the root server configuration node that you wish to remove.
3. Right-click on the server configuration node, and choose **Delete**.
4. Click **OK**.

The Server Explorer updates to remove the deleted server configuration.

## Linking Views with Selections

Automatic refresh of the view contents on selection changes in other views may be undesirable as it can considerably slow down the work flow. You may also want views to show the same content during the whole session such as change requests in process-related folders.

1. Take one of the following actions:
  - Open one of the StarTeam perspectives; or



- Open a single StarTeam view.
2. Click **Link with Selection** in the view's toolbar.
  3. To unlink, click **Link with Selection** again.

## Exporting a Set of Team Preferences

1. Choose **File > Export** from the main menu. The **Export** wizard opens.
2. Expand **Team**, and select **Team project set** from the list.
3. Click **Next**. The **Export a Team Project Set** page of the wizard opens.
4. Take the following actions:
  - a) Select a project or multiple projects that you wish to include in the set of team preferences.
  - b) Choose an export location for the project set file.
  - c) Click **Finish**.

Eclipse creates the `projectSet.psf` file and exports it to the location specified.

## Importing a Set of Team Preferences

If you work in a team environment, you can import a set of team preferences.

1. Choose **File > Import** from the main menu. The **Import** wizard opens.
2. Expand **Team**, and select **Team project set** from the list.
3. Click **Next**. The **Import a Team Project Set** page of the wizard opens.
4. Take the following actions to complete the task:
  - Select the `*.psf` file name to import. The application automatically names the file, `projectSet.psf`.
  - If the `*.psf` file contains more than one project, you can optionally check the option to **Create a working set containing the imported projects**, and enter a name for the working set.
  - Click **Finish**.
5. Open the Server Explorer.
6. Right-click on the server configuration connection and choose **Log On**.
7. Enter your log on credentials for the server configuration, and click **OK**.

## Displaying Additional Fields

1. Open the Change Request, Requirement, Task, Topic, or Audit Log view.
2. The **Show Fields** dialog box displays two lists. The **Available Fields** list contains all the fields that could be displayed as column headers but are not currently displayed. The **Show These Fields in This Order** list displays all the fields that are currently displayed.

Do any combination of the following:

**Display additional fields** Select the fields to display as the column headers from the **Available fields** list. Then click **Add**.

**Stop displaying fields** Select the fields to be removed from the **Show these fields in this order** list. Then click **Remove**.

**Change the order of the fields** Drag each field name to the desired location in the **Show these fields in this order** list.

3. Click **OK**.



**Tip:** Double-clicking a field name moves it from one list box to the other. The **Show Fields** dialog box initially displays the most commonly used fields. Check the **Show Advanced Fields** check box to select from a complete list of the available fields.

## Setting Background Operations

1. Choose **Window > Preferences** from the main menu. The **Preferences** dialog box opens.
2. Select **General**.
3. Check **Always run in background**.
4. Click **OK**.

## Viewing Background Operations

1. Choose **Window > Show View > Other** from the main menu. The **Show View** dialog box opens.
2. Expand **General** and select **Progress**.
3. Click **OK**. The **Progress** view opens.



**Note:** To cancel background operations click **Stop** to the right of the progress bar.

## Refreshing Server Configurations

If your StarTeam server goes down, you can still work with other server configurations checking out files from various server configurations in parallel.

1. Open the Server Explorer.
2. Select the server configuration with a *Pending* status.
3. Right-click on the server configuration and choose **Refresh**.

If it is available, the StarTeam Eclipse Plugin reconnects to the selected server configuration.

## Compare Revisions

You can use the **Compare With** context menu commands for, among other things, comparing local versions of files and folders to those in the StarTeam Server repository.

Using the **Compare With** context menu commands, you can complete the following actions:

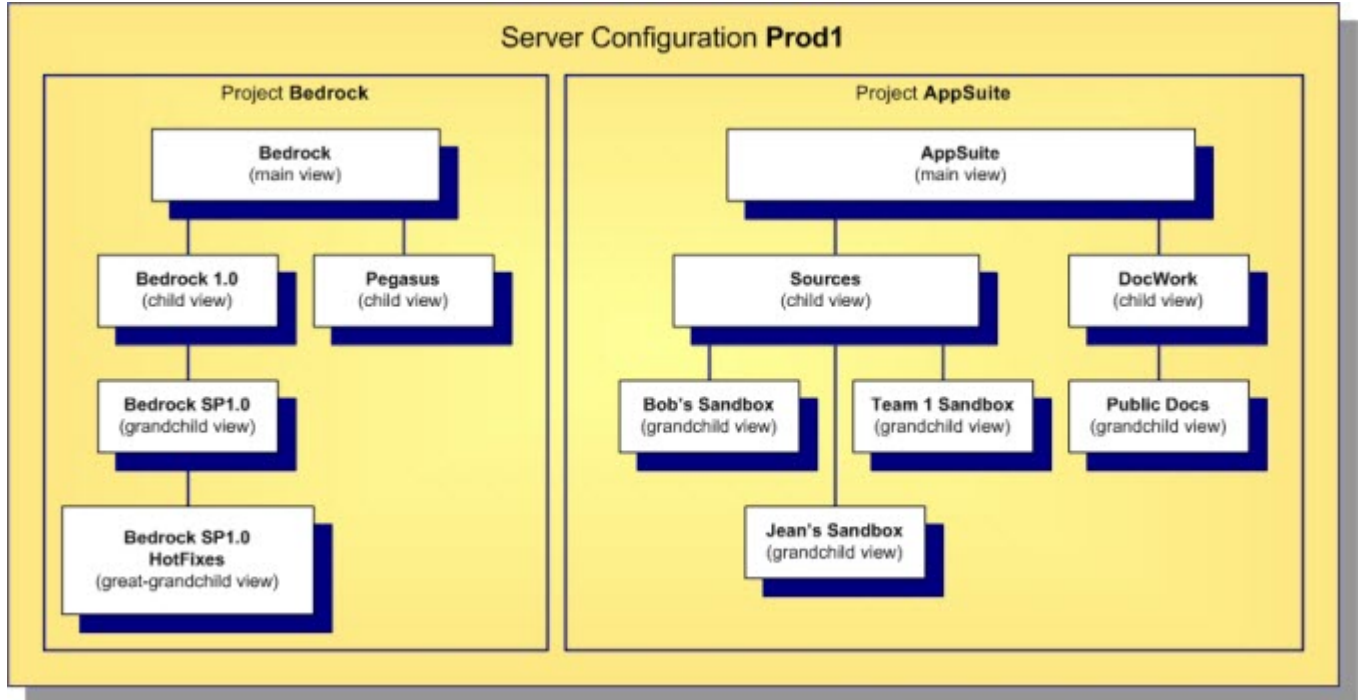
- Compare your local file to the latest tip revision stored in the repository.
- Compare a local folder or file to a labeled version stored in the repository.
- Compare a local folder or file to a copy in a different StarTeam view stored in the repository.
- Compare a local folder or file to a copy of that same item determined by a date.
- Compare a local file to a specific revision stored in the repository.
- Select two files or two folders and compare their properties.

## StarTeam Basics

The topics in this section provide an overview of StarTeam concepts.


# Containers

StarTeam Server configurations, projects, and views are *containers* that allow you to organize artifacts based on application, module, business unit, or other criteria. These three basic containers are illustrated in the diagram below:



## Server Configurations

A *server configuration* is also referred to as a repository or as an instance. All files, change requests, and other artifacts that can be interrelated and managed as a whole reside in the same configuration.

 **Note:** Throughout the documentation, the terms *server configuration* and *server* are also used interchangeably. This is because each server configuration is often deployed on its own server machine, managed by its own StarTeam Server process. However, be aware that StarTeam allows multiple server configurations and server processes on a single machine, so the server configuration-to-machine relationship does not have to be one-to-one.

## Projects

Within a server configuration, artifacts are organized into *projects*, which group and manage related items hierarchically in a set of folders. Creating a project allows you to put files under version control, set requirements, track change requests, manage tasks, audit user actions, and discuss the project. Each project has at least one view, called the initial or root view. For example, a project for a software product might include files on the product's functional specifications, marketing requirements, source code, and test suites, all stored in separate folders in the initial view. As the product progresses from one release to another, additional views of these folders can be created. One view could represent the 1.0 version of the product, while a second view represents the 2.0 version, and so on.

Before a server configuration can be used, at least one project must be created. A server configuration can hold multiple projects, each oriented to the lifecycle needs of a specific team, application, or component. The configuration in the diagram above has two projects: *BedRock*, perhaps for foundation components, and *AppSuite*, which could be used for applications belonging to a common suite.

## Views

Each project consists of one or more *views*. Think of a view as a *subproject*. It is a subset of the project's contents that support a specific activity. Every project automatically receives a *main view* through which folders, files, and other objects can be organized. Additional *child views* can be created to represent subsets of the main project information, historic snapshots of project information, or work areas for new development work. StarTeam provides a wide range of options for view creation to support a wide range of development scenarios.

## Workspaces

In addition to the three basis containers, StarTeam supports a client-side container called a *workspace*. A workspace is a folder hierarchy located on your computer or in your personal directory on a shared file server. However, the project does not have to exactly match your working folder and its child folders. For example, you may omit child folders in the working folder from a project or copy only specific child folders in an existing project to the working folder. When you add or check in files, the application copies the files from the working folder into the repository. When you check files out, the application copies the files from the repository into the working folder.

In addition to providing a well-defined area for check-in/check-out operations, a workspace allows StarTeam to compute the status of each file: which ones have been modified since they were checked-out, which ones are out-of-date, and so forth.

## Artifacts

A typical software development lifecycle requires the development, evolution, and management of things other than source files such as requirements, models, graphics, change requests, schedules, tests, and so on. The term *artifact* refers to the generalization of objects that can be versioned, branched, merged, etc. StarTeam supports non-file artifact types directly, providing type-specific behavior for storage, versioning, merging and so forth. All artifacts are versioned, and some are branchable.

The built-in artifact types supported by StarTeam are summarized below:

<b>Folder</b>	Every view has one <i>root folder</i> , which typically has a tree of subfolders beneath it. Folders are patterned from the file system concept of directories. In many cases, you will want to create StarTeam folders that mirror specific directory structures. However, StarTeam folders can hold any kind of artifact, not just files. This concept may seem strange at first, but when you discover that you can organize change requests, tasks, and other non-file artifacts the same way you organize files, you will find this feature very powerful. Folders can branch, allowing the same folder to have different properties in each branch.
<b>File</b>	StarTeam allows you to store any kind of file: text or binary, authored or generated, small or very large. A few more features are provided for text files such as <i>keyword expansion</i> and <i>EOL conversion</i> , but all file types otherwise are treated identically. StarTeam allows single file revisions larger than 4GB. Files are branchable, allowing parallel version streams and merging.
<b>Change Request</b>	A <i>change request</i> (CR) is a general artifact that can represent a defect, enhancement request, or another reason for a software change. Because CRs are often the center of change management, the CR type is frequently extended with custom fields, custom GUI forms, and workflow rules. CRs can branch, allowing parallel modifications to the same CR for separate activities such as fixing the same defect in multiple releases. Using integration tools, you can import CRs from and keep them synchronized with other defect management systems.
<b>Task</b>	StarTeam <i>tasks</i> are modeled after project management tasks: they can be arranged hierarchically to represent task decomposition, they can be connected with predecessor/successor relationships, and they can be updated with progress units known as <i>work</i>

*records*. You can import tasks from a project management system such as Microsoft Project, update and maintain them via StarTeam, and then synchronize them back to the original project source. In StarTeam, tasks are versioned but they do not branch.

<b>Topic</b>	A <i>topic</i> is very similar to a newsgroup message. Like newsgroup messages, topics can be organized into conversation threads. Because topics are artifacts, they are versioned (but not branched) and are stored in the repository with other artifacts. This allows you to capture more application lifecycle knowledge such as important discussions related to a design decision or a requirement approval.
<b>Requirement</b>	If you do not have a formal requirements management (RM) tool, StarTeam requirements provide a convenient, lightweight artifact with which requirements can be captured. Requirements can be arranged hierarchically to represent decomposition, and they can be linked to other artifacts. Since requirements are independently-versioned artifacts, they are more accessible than requirements buried in documents, which are versioned at the whole-document level. If you use a requirements management system such as Caliber, those “formal” requirements can be imported as StarTeam requirements and organized together with other lifecycle artifacts. We provide integration tools to import, synchronize, and even link artifacts between StarTeam and Caliber. Requirements do not branch.
<b>Audit</b>	An <i>audit</i> is a read-only <i>change event</i> artifact that is automatically generated for other artifact changes: add, modify, delete, move, label attach, link, etc. Because audits are automatically generated and immutable, they are not really artifacts per se, but StarTeam allows you to access them with similar GUI and SDK techniques as other artifacts, so you can think of them as read-only artifacts. The generation of audits and the length of time that they are retained are configurable.

These artifacts are all bundled with StarTeam, however you are not obligated to use them all. The code for each artifact type is encapsulated in a dynamically-loaded plug-in module called a *server-side component* (SSC). Each SSC is a code library suffixed with `.ssc` that resides in the StarTeam Server’s installation directory. If you rename an `.ssc` module before the server starts, the corresponding artifact type will not be used. For example, if you want to use StarTeam as a VCS only, just rename all `*.ssc` modules except for `file.ssc`.



**Note:** You always get folders, so there is no `.ssc` module for it. Also, we recommend you keep `audit.ssc` due to the value of the *change log* represented by audit artifacts.

## Artifacts Versus Items

The difference between artifacts and items is that you can only access and update artifacts through items. Since StarTeam blends item and artifact properties into a single object (both graphically and in the SDK), you may think of them as a single concept. Although either term works equally well in most cases, we usually use the term *item* when we mean to include folder/view context that items add to artifacts. When the context is not important to the discussion, we use the term *artifact*.

The most important things to know about items are summarized below:

<b>Artifacts can only be accessed through items</b>	With StarTeam, you can only fetch or update an artifact by directing your request to a specific item. There are no commands to directly access an artifact independent of an item. This means that all artifact access is influenced by the context of the associated item such as its parent folder and the view in which it lives.
<b>Items form folder trees</b>	<i>Paths</i> are formed by each view’s <i>item tree</i> . This means that folder artifacts do not define their contents. Instead, what appears inside a folder is determined by the items that refer to the folder’s item as their parent. You don’t really move artifacts—you move items. Moving an item from one folder to another causes the item’s parent to be modified—the artifact referred to by the item isn’t touched. Under the hood, items are versioned

similarly to artifacts. This means that changes such as moving an item to a new folder really creates a new item revision, causing the previous item to become historic.

**Items facilitate sharing** Items allow an artifact to appear in multiple folders, views, and projects. To make an artifact, including its entire history, appear in a new location, we only have to create a new item, which is pretty cheap. Sharing is analogous to “hard links” used in UNIX file systems.

**Items influence version behavior** An item has properties that control what artifact revision is referenced and how updates through the item are handled. Items store an OID that determines what artifact branch is referenced. An item also stores a *configuration timestamp* to indicate whether it *floats* to the tip revision or is *pinned* to a specific revision of the referenced branch. An item’s **branch-on-change** (BOC) flag indicates if the referenced artifact should branch when modified through the item. For example, if an item currently refers to artifact revision 1.7, and **branch-on-change** is true, and an update is directed at the item, the artifact is branched by storing the updates to a new revision identified as 1.7.1.0. Additionally, the item is modified to point to the new branch (since it has a new OID), and its BOC flag is set to `false`. Note that **branch-on-change** cannot be true for items that point to artifacts that can’t branch (such as topics). Also, an item with **branch-on-change** equal to false and a pinned configuration timestamp is read-only because we can’t update a historic revision and we can’t start a new branch!

**Items create promotion trees** This is an advanced concept, so we’ll just touch on it briefly here. Items that are shared to a new location “remember” the item from which they were shared. This “share parent” relationship is different than the “containment parent” relationship that forms item paths. It facilitates a concept called *automatic promotion*.

## Folders

The project or server administrator usually creates projects and project views. If you are a typical user, you routinely open a particular project view and manage *your* folders and their contents, such as files and change requests. Managing application folders is very similar to managing a project. You can create folders, delete folders, and modify their properties—if you have the correct access rights.

### Folder Hierarchy

When you create projects, you typically select locations on your workstations as the working folders for those projects. The working folder designated for a project also becomes the working folder for the project’s root view and for the root folder in that view’s folder hierarchy.

StarTeam treats folders as both containers and items. You can group items within a project view by placing them into folders. For example, a folder named *Source Code* can contain source code files and requested changes to those files. You can create folders automatically when you create a project, or add folders after you create the project. Project or server administrators (or team leads – this all depends on your organization) usually create projects, but anyone can create projects if they have the correct access rights. See your server administrator if you have questions regarding the access rights assigned to you.

When you create a project, StarTeam automatically creates the parent or root folder for that project at the same time. It is actually the root folder of the project’s root (or initial) view. The project, view, and this root folder initially have the same name (although those names can be changed).

Usually, the user who creates a project sets up a hierarchy of folders on a workstation before creating the project. The user designates the root folder of that hierarchy as the project’s working folder. Then the application can automatically create an application folder for each of the child folders in the hierarchy. The child folder becomes the application folder’s working folder.

If child application folders are created at the time the project is created, then:

- The application folders’ working folders were part of an existing hierarchy on the project creator’s workstation.

- Their names are the same as the names of their working folders, but they can be changed later.
- Their working folders remain hierarchically connected to the root folder's working folder. That is, if you change the path to the root folder's working folder, you also change the path to this folder (unless you manually set an absolute path for these working folders). In other words, the application stores a relative path to each child folder.

One of the most important properties to notice about your folder is its working folder. You will need to know where on your workstation the application will copy file revisions that you check out so that you locate those revisions as needed for modifications. A number of other operations can be performed on folders, such as moving a folder or changing its branching behavior.

A working folder is a property of the folder and represents the actual location on your workstation where StarTeam saves files that you check-out. Despite the fact that these are both called folders, the working folder and the folder are not identical. Their differentiating characteristics include:

- The path to the working folder can be totally different from the path within the application to the application folder.
- An application folder is an object controlled from within the application. The data associated with this folder is stored in the database that stores all the project data.
- A working folder is an object controlled by your operating system. It stores files that are checked out from the application.

A project, its root view, and the root folder of the root view all have the same working folder. For additional views, each view and its root folder have the same working folder.

The working folder for the view/root folder always has an absolute path (starting with the drive letter and specifically naming the folders at subsequent levels until you reach the working folder itself).

If you look at the properties for the root folder, you will see that the working folder is the same. However, it is displayed in the **Complete Working Folder Path** display box instead of the **Default** field. Since you can only change the working folder at the view level, all of the fields for the root folder's working folder are always disabled.


For the child folders that were created at the same time as the project, the application stores the path to each working folder as a relative path.

## Folders and Views

You can add new folders and Not-in-View (NIV) folders to views. A NIV folder is a folder on your local disk that does not map to a folder in the StarTeam repository. A NIV folder is displayed as a white folder with a black, dotted border. NIV folders (as with NIV files) do not necessarily need to be added to a view, but you may choose to do so if you just created it and want it to be part of the view. However, if the folder is NIV because someone else deleted it from the view, you may need to delete it from your working folder.

If you add a new folder to a view, its working folder can be any of the following:


- Any folder on your workstation specified by you.
- A non-existing working folder specified by you and created by the application on your workstation. If the existing folder has child folders, one or more of them can also be added to the view.
- A child of the parent application folder's working folder. If you do not specify a working folder, the application appends the new folder's name to its parent's complete working folder path.

 **Note:** If the parent folder's working folder path length exceeds the operating system's maximum working folder path length of 254 characters (including (\) backslashes), the application does not allow you to create the new working folder. Also, you cannot add a folder to a view if the parent folder is read-only. The newly added folder assumes the parent folder's behavior, with a few exceptions. For example, the child folder might have the **Branch On Change** check box disabled because it makes no sense for this folder to branch.

## New Folders

You can easily add folders to a project view. When a new folder is added:

- The working folder for the new application folder does not have to belong to the same hierarchy as the other application folders' working folders. However, if it uses the same drive letter as the root folder's working folder, its path is stored as a relative path based on the path to the working folder of its parent folder in the hierarchy.
- Its name can be different from the name of its working folder.
- If the new working folder has child folders, a folder can be created for each of the children. Essentially, the newly added folder becomes the root of a new branch of folders. The application folders created for the child folders take the names of their working folders—at least initially. The working folders retain their relationship to the working folder that is the root of their hierarchy (that is, the working folder for the newly added folder). If you change the path to the newly added folder's working folder, you also change the path to these working folders (unless you manually set an absolute path for these working folders).

StarTeam indicates folders on disk that do not map to a StarTeam folder with a **Not-In-View** icon . This indicates that you do not have the folder in the project view.

## Existing Folders

You can add more folders to a view by moving them or sharing them from other views on the same server configuration. When a folder is moved or shared, it either keeps its absolute path or its relative path and is applied to its new parent folder. When a moved folder's path is relative, it usually ends up with a different working folder than it previously had. When a shared folder's path is relative, the shared folder has a different working folder in each location.



**Note:** The application does not allow you to create a working folder if a shared or moved folder's new working folder path exceeds the operating system's maximum working folder path length of 254 characters (including (\) backslashes).

Both the current view and the view from which the folders are moved or shared must use the same server configuration—and, therefore the same database and repository.

## Files

To place a file under version control, it must be added to a folder in a StarTeam project view, which stores a copy of the file in the StarTeam repository. After the file has been added to StarTeam, you and other members of your team can check it out, revise it, and check in new revisions, while StarTeam maintains information on all revisions of the file. Note that all check-ins in StarTeam are atomic.

When checking out a file revision, you should verify that you have the *tip* or latest version of the file. Doing this ensures that the file you see contains the latest changes. If you intend to modify the file, you should check it out with an exclusive lock, to indicate to others that you are working on it.

When you check a file in, StarTeam records the file changes as a new revision. As part of the check-in process, you can remove the lock, notifying others that the file is available, or maintain the lock, showing that you intend to continue working on the file. If two team members change the same text file simultaneously or if one member changes an outdated file, StarTeam contains a merge option that allows the file changes to be combined so that no work is lost. In such cases, StarTeam assigns a *Merge* status to the file.



**Note:** The SDK, StarTeam Server, and most clients support files larger than 4 GB. If you plan on taking advantage of large file support, you should upgrade all users to the current StarTeam version. Large file sizes are not compatible with older StarTeam versions.



**Note:** Using the StarTeam Eclipse Plugin, operations for files are surfaced using the:

- **StarTeam** main menu commands.
- Context menus provided in the various StarTeam views.



- StarTeam-specific toolbar buttons.
- **Team** context menu.
- **Team Synchronizing** perspective.

### Files Under Version Control

If a file resides in the working folder of an application folder, you can add that file to the application folder. This operation places that file under version control. A copy of the working file becomes the first revision of that file stored in the repository. If the working file is deleted later, the data is not lost because a copy exists in the repository. The application creates a new revision of this file in the repository every time you check the file in.

Every time you check a file revision out, its contents are copied to a working folder. Checking out a revision also ensures that you have the tip or a specific revision to work on. For example, you may need a team member's most recent changes to a file, or you may have deleted the working file from your hard drive and now need another copy.

The application enables you to label the tip revisions of every item within a view. For example, when the project reaches a particular milestone (such as beta), you might give the view's items a label, called a view label. Then you can configure the view to return to the way it was at the time the label was applied, check out revisions as a group using that label, create a new view based on the label, or assign the label to a promotion state.

The application also provides revision or version labels. You can label one or more revisions as you check them in or by applying the label to each of the revisions using the **Labels** command on the File menu. StarTeam makes it easy to check out those files as a group using the label. A file revision can have any number of labels. However, no two revisions of the same file in the same view can have the same label.

### Recommendations for Working with Files Under Version Control

Here are some recommendations about using files under version control:

- To let other team members know that you intend to make changes to a file, change the lock status to *exclusive* as part of the check-out procedure.
- As part of the check-in process, you can notify others both that you are finished making your changes to the file and that it is available for them to check out by removing the lock status.
- If you intend to continue making changes to the file but still want to check it in for backup purposes, keep the file locked.
- If two team members change the same text file simultaneously or if one member changes an outdated file, you can use the merge option to combine the changes in these files so no work is lost. In such cases, the application gives the file a *Merge* status.
- To prevent yourself from changing a file that you have not locked, select **Window > Preferences > Mark Unlocked Working Files Read-only** the **Mark Unlocked Working Files Read-only** preference. Then, if you check out a file that you have not locked, the working copy becomes read-only.


## Change Requests

The *change request* component provides a defect tracking system that allows you to record defects in products, projects, or services and suggest possible enhancements. A change request is a request to change something within the scope of a project. For example, you might suggest a product enhancement or request a fix for an error or problem. To use the change request tracking system effectively, you need to understand the model on which it is based.

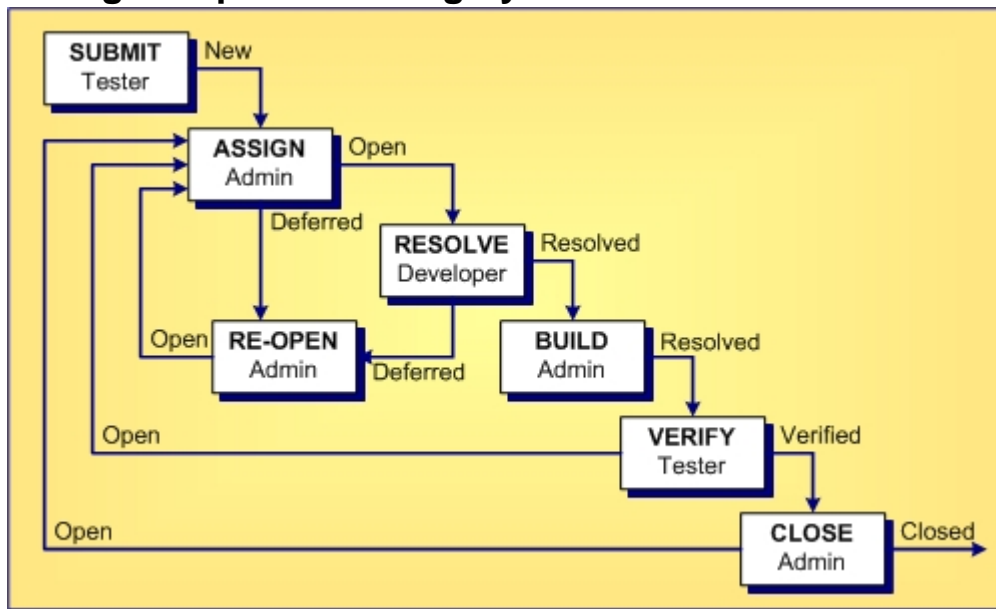
The change request component allows you to:

- Attach change requests to any folder. In the application, change requests can be attached to any project folder or shared among folders or other views in the same server configuration. You can also link a change request to any other item, such as a file. In many other defect tracking systems, a change request can be associated only with a project, even though it requires modification of a particular file.

- Save time when updating change requests. When you check in a file or group of files, you can indicate the change requests that are fixed by the files being checked in. This feature saves the time required to change the status of each change request separately.
- Make only appropriate status changes. When you create a change request, the status options are *New*, *Open*, *Deferred* or a resolution. The resolutions are *Cannot Reproduce*, *As Designed*, *Fixed*, *Documented*, and *Is Duplicate*. After resolution, a change request can only be verified or reopened. After verification, a change request can only be closed or reopened.
- Benefit from automatic changes based on the status of the change request. The application automatically changes the person responsible to coincide with the current status of the change request. When a change request is resolved, the responsibility for the change request automatically reverts to the person who entered the change request, who is usually the best person to verify its resolution. When a change request is reopened after being resolved, the responsibility is automatically set to the user who resolved it. If desired, you can override these automatic changes and make another person responsible.
- Base change requests on the build in which the change request is resolved. When a change request receives a **Fixed** or **Documented** status, the value of its **Addressed In Build** field becomes **Next Build**. When that build label is created, the application replaces **Next Build** with the name of the build label, letting testers know the build to use when verifying change requests.

 **Note:** This help system explains how to use the standard property dialog to create and edit change requests. Depending on how your team has set up the application, you may see a different dialog called an alternate property editor (APE). Even if you use the standard property dialog for change requests, your company or team leader may implement change request guidelines that differ from those discussed in this help system.

## Change Request Tracking System Model



The above diagram and steps show the change request tracking process. The boxes represent the steps taken from the time that the change request is submitted until the time it is closed. Each box indicates an action and the team member most likely to be responsible for performing this action. The arrows show the status of the change request at the time of each step.

The change request tracking system consists of the following steps:

**Step 1** A team member creates a new change request that does either of the following:

- Summarizes a problem with the product and lists the steps taken to reproduce the problem.

- Suggests an enhancement to the product.

This change request has a status of `New`.

**Step 2** Another person, such as an administrator or team leader, decides whether to fix the problem or add the suggested enhancement to the product. This person can:

- Set the status of the change request to `Open`, and assign a team member to resolve it.
- Set the status of the change request to `Deferred` because it is worthwhile but will not be done at this time.
- Set the status of the change request to `Is Duplicate` because this is not the first time it has been submitted. If desired, a link can be created between a change request and the original submission so that you can track the change request along with the original submission.
- Set the status of the change request to `As Designed` because the product is supposed to work this way, meaning there is no defect.

Change requests with an `Open` status go to step 3.

**Step 3** The person assigned to resolving the change request changes the status of the change request to `In Progress`. Later on, after this person finishes examining the change request, he or she changes the status to one of the following:

- `Fixed`
- `Documented`
- `Cannot Reproduce`

**Step 4** Next, a team member (usually a tester or quality assurance engineer) verifies the change request. For example, a test case may be developed to determine if the problem is really fixed, documented or not reproducible and changes the status to one of the following verified statuses:

- `Verified As Designed`
- `Verified Cannot Reproduce`
- `Verified Documented`
- `Verified Fixed`
- `Verified Is Duplicate`

**Step 5** Finally, another team member changes the status to `Closed`. This person may then perform activities related to closing the change request, such as retesting the change request before closing it or adding it to a report to be included in the next release of the product.

In most of the above steps, the change request can be reopened and reprocessed.

## Built-in Workflow for Change Requests

StarTeam has a built-in workflow for change requests that automatically sets many of the values associated with change requests. This built-in workflow determines these settings based on the setting of the **Status** field for the change request.

You cannot add additional settings to the **Status** field. However, you can rename them to better suit preferences set by your organization. For example, your organization may prefer to change the name of the value `New` to `New Change Request`.

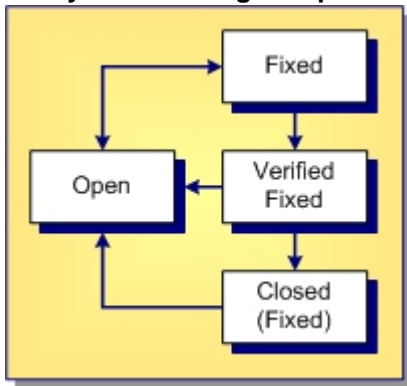
When you alter the status of a change request, the built-in workflow automatically selects the appropriate properties associated with the change in status.

After selecting `New`, `Open`, or `In Progress`, six new statuses display in the **Status** list. These statuses, which are associated with the status you selected, are:

- `Deferred`

- Cannot Reproduce
- As Designed
- Fixed
- Documented
- Is Duplicate

### Lifecycle for Change Requests



The above diagram shows the lifecycle for a change request with an initial status of **Open**. The status was then set to **Fixed**. After this setting, the built-in workflow added an additional status field of **Verified Fixed**. Finally, the change request was closed, meaning its status was set to **Closed (Fixed)**.

The diagram also shows that a change request can be reopened at any stage in its lifecycle because the arrows leading from each of the three fixed statuses can lead back to the **Open** status at any time.

### Summary of the Change Request Automatic Workflow

The following summarizes the steps used in processing change requests as explained in this topic. It includes the automatic workflow changes the application makes to change requests based on their statuses.

**Submit** Anyone (usually a tester or quality assurance engineer) can submit a change request.

**Process:** Select the **Change Request** tab. Then select **New** from the **Change Request** or context menu.

A change request has the following default properties (which you can change if necessary).

**Status:** New

**Severity:** Low

**Priority:** Not prioritized

**Type:** Defect

**Platform:** All

**Last Build Tested:** Current build label

**Entered by:** Person currently logged on to the application

Many other fields are initially blank. Some team leaders prefer to have all change requests submitted at the root folder. They use drag-and-drop to move the change requests to the appropriate child folders.

**Assign\*** **Process:** The team leader finds all new change requests and does one of the following:

- Opens the change request and assigns it to a developer, help writer, or other appropriate team member.
- Defers the change request until a later date, perhaps the next release of the product.
- Specifies that the change request is `As Designed` and not to be fixed. If the change request status is `Open`, no automatic changes occur. If the change request status is `Deferred` or `As Designed`, then `Addressed in Build` is disabled and the responsibility is assigned to the user who created the change request.

**Resolve Process:** Users find the `Open` or `In Progress` change requests assigned to them, and do one of the following for each request:

- Resolve the problem in the system and update the properties of the change request. (The statuses that indicate that a change request has been resolved are `Cannot Reproduce`, `As Designed`, `Fixed`, `Documented`, or `Is Duplicate`.)
- Defer the change request until a later date, perhaps the next release of the product. Your team leader may prefer that you do not defer change requests.
- If the change request status is one of the possible resolution statuses, then **Addressed in Build** becomes `Next Build` for `Fixed` and `Documented` statuses. It becomes disabled for other statuses. By default, the responsibility is assigned to the person who submitted the change request, who is expected to verify the resolution.
- If the change request status is `Deferred`, then **Addressed in Build** is disabled and the responsibility is assigned, by default, to the user who created the change request.

## Build

Who builds the project? The project view may have a formal or informal build process. However, at some point, all the files, and so on currently in the view receive that build label. It is usually applied to the source code files, and so on that were compiled (and may need to be changed) rather than to the executable files that result from the build.

Effect on change requests: For any resolved change request that has `Next Build` as the setting for its **Addressed In Build** property, `Next Build` is replaced with the next build label that is created.



**Note:** If a new build label is based on a past configuration (rather than the current configuration), it has no effect on the `Addressed In Build` property.

If a change request has not branched in its current location, `Next Build` may be replaced with a build label from another view. For example, suppose you create a branching child view or share a folder from one view to another. Suppose that `Next Build` is the value of some change request's `Addressed In Build` property and that change request has not branched. When a build label is created in the source view, `Next Build` is replaced with the name of that build label, regardless of the location.

## Verify\*

The person who submitted the change request (usually a tester or quality assurance engineer) verifies a resolution.

**Process:** Install the build in which the resolution is to be verified and determine whether the change request has been resolved correctly. Do one of the following:

- Verify the change request, marking it as `Verified Cannot Reproduce`, `Verified As Designed`, `Verified Fixed`, `Verified Documented`, or `Verified Is Duplicate`.
- Reopen the change request and update the setting for **Last Build Tested**.
- If the change request status is `Verified`, no automatic changes occur.
- If the change request status is `Open`, **Addressed in Build** is blank. If the change request has changed from resolved to **Open**, the user who changed the status to `Fixed` or `Documented` becomes responsible.

**Close\*** Usually the team leader closes the change request.

**Process:** The team leader does one of the following:

- Reviews and closes the verified change request.
- Reopens the change request.
- If the change request status is `Closed`, then no automatic changes occur.
- If the change request status is `Open`, then **Addressed in Build** is blank. If the change request has changed from `resolved` to `Open`, the user who changed the status to `Fixed` or `Documented` becomes responsible.
- If the status of a change request changes from `Verified` to `Open`, the user who changed the status to `Fixed` or `Documented` becomes responsible and **Addressed in Build** is blank.

\*Changes in status can result in automatic changes to other properties.

## Requirements

Requirements are supported for the Enterprise Advantage license and display in the **Requirement** tab of the upper pane in the clients. With the requirement component, you can create requirements within the application and show the dependencies among them. For example, if one requirement must be fulfilled before a second requirement can be fulfilled, the first can be made a child of the second. If your company enforces process rules, the requirements you establish can also be used to drive the development process. Administrators and other authorized users can publish requirements from Caliber to StarTeam using Publisher to StarTeam, which is delivered with Caliber.

### Requirement Characteristics

The requirements in the upper pane have the following characteristics:

- They are attached to the folder selected from the folder hierarchy.
- They match the filter selected from the **Filter** list.
- They match the depth specified by **All Descendants**.



**Note:** You can click the **All descendants** button on the **Requirements** view toolbar.



**Note:** Icons display to the left of a requirement in the upper pane to indicate its status and whether you have read the latest revision.

### How Requirements Can Help

By using a requirements-driven development processes, companies can prevent consuming, costly misunderstandings and shorten time to market. To accomplish this, you can use the StarTeam built-in requirement component as your basic tool, or publish complex requirements to StarTeam from Caliber. Using requirements enables business analysts, managers, developers, QA staff, and others to:

- Organize business, user, and functional requirements in a hierarchical format.
- Indicate the dependencies among requirements.
- See all layers of requirements at all times.
- Prioritize requirements by importance.
- Identify the impact of changes to requirements.
- Use requirements to estimate work.
- Identify the person creating the requirement.
- Notify those who will be responsible for fulfilling the requirements.
- Track the requirement life-cycle from submitted to completed or rejected.

- Provide requirements with a context by linking them to files, change requests, and topics.

## Tasks

The *task* component allows the creation of task lists and work assignments. As a standalone, the task component is very useful for managing a project. It allows team members to indicate who should do what and when, see current task status, estimate hours required to complete a task, record hours spent completing the task, and compare estimated to actual times. Because the application contains both a version control system and a change request system, it also allows tasks to be linked to the files and product defects or suggestions with which they are associated.

The task component can be used independently or interoperate with data from Microsoft Project. It can display tasks in a tree format, which clearly shows the relationship between tasks and subtasks, or in a list format, which allows tasks to be sorted, grouped, or queried, or specific fields to be selected for display. To improve efficiency, each task displays icons that identify its status, priority, milestone, and need for attention. For information about interoperating with Microsoft Project, see the *StarTeam Microsoft Project Integration User's Guide*.

With the StarTeam task component, you can create an individual task or a summary task that has a set of subtasks. It is recommended that you plan tasks before entering them because:

- A task that has even one subtask cannot have work records added to it, although work records can be added to subtasks. The application assumes that the name of the task indicates a goal, perhaps a milestone, that will be reached when the subtasks are completed.
- After a work record has been added to a task, you cannot create subtasks for it.



**Note:** Regardless of whether work records can be added to a task, you should assign the responsibility for its completion to a specific team member. If work records can be added to a task, you should also estimate how long the task should take.

## Topics

*Topics* are threaded conversations, that is, a series of messages that indicate how the messages are related. Each series of messages forms a tree with the initial message at its root. The topic component provides threaded conversations that you can place in specific project folders and link to specific project items. For example, you can link a topic to the change requests and file revisions that result from the topic discussion.

The **Topic** view consists of topics and a series of responses to each topic. A series of topic trees are eventually formed, each of which consists of a root topic and its responses. The topic tree resembles a conversation that may go on among several people. In the client, this is called a threaded conversation because a topic and its responses are threaded together, starting with the root topic. By reading each response in a thread, one after the other, and the responses to those responses, you can see how the discussion has evolved. A number of other operations can be performed on topics or responses such as moving or sharing them.

### Historical Value of Topics

Topics can raise general questions about the project or start very specific discussions about issues, such as feature implementation. While the responses can lead to resolution of these issues, the historical value of these conversations to the project can be even more significant. Future team members can:

- Reassess decisions more capably.
- Avoid retrying solutions that were previously found faulty.
- Understand why a particular solution to a problem became necessary and, therefore, not replace that solution with one that does not meet all the necessary criteria.

## How Topics Can Help

Any type of threaded messaging improves teamwork on product development. However, because StarTeam has tightly integrated components, it enables team members to:

- Search topics and responses for specific words or phrases.
- Sort topics and responses.
- Filter topics and responses.
- View relationships between topics and their responses.
- Move and share topics (from the tree format).
- Link topics directly to folders or other items, such as change requests.
- Ask questions and quickly receive input while working on a file.
- Attach notes to a topic explaining why a particular method was used.
- Point out aspects of the project that may need to change in a later release.

## Links: Internal and External

A link is a connection between two folders, two items, or a folder and an item on the same server, or on two different servers (called *External Links*). Creating links can be quite useful. For example, linking a file to a change request allows you to mark it as fixed when you check in the edited file. By linking files to the requirements document that the files fulfill, you can easily refer to or update the document.

In addition, linking files to change requests enables you to mark the change requests as fixed when you check in the corresponding files. In turn, if you link each set of files to the requirements document that the files fulfill, you can easily refer to or update the document.

A link does not provide a connection to a single share (or reference), but to all related shares and branches of an item. Links are not affected by any item operations, such as branching, moving, sharing, and so on. By default, a link connects the tip revisions of the linked pair.

Links can either be pinned or floating:

**Pinned** You can lock the link to the tip revision. The context menu in the link view enables you to pin links to the source or target items or both.

**Floating** You allow the link source or target to change from tip revision to tip revision as new revisions are created. The context menu in the link view enables you to float links to the source or target items or both.

Links, as with all other items, have context menus in their views which allow you access to more information about the item.

## Labels

In version control, the term `label` corresponds to the act of attaching a view or revision label (name) to one or more folders/items. StarTeam enables you to create two types of labels:

**View labels** Are automatically and immediately attached to all folders and items in a view at the time you create the view label. View labels have multiple purposes, but you primarily use them to place a *time stamp* on the entire contents of the view and as *build labels*. When you roll back the view to that label, you see everything that existed at that point in time—unless the label has been adjusted. You can create a view label for a specific point in time or as a copy of another existing view label. Unless the view label is frozen, you can adjust it to include or exclude some folders and items by attaching or detaching view labels. You can also move a view label from one revision to another.

**Revision labels** Are not attached automatically to any item in the view. Instead, they are used to designate a set of folders or items within a view. For example, you might want to label a group of files that



should be checked in and out together. After you create a revision label, you attach specific items, building it up to reflect a specific set that is typically a small subset of the view. StarTeam can automatically attach new file revisions to a revision label at check-in time if you like.

## About Labels

You can attach a label to any type of StarTeam item, including folders, files, requirements, change requests, tasks, topics, and audit entries. Any item can have more than one label. However, no two revisions of the same item can have the same label at the same time.

Every label is unique within its view. That is, no view label can have the same name as any other view label, no revision label can have the same name as any other revision label, and no view label and revision label can have the same name. Each view has its own set of labels. This also means that every label has a *name* that must be unique from other labels belonging to the same view. Each label also has a *description* that helps users understand the purpose of the label.

You can manually attach or detach both view labels and revision labels to or from a folder or item. In addition, you can use either type of label to identify a file when it is checked out. When you check a file in, you can attach and create a revision label for that file or attach an existing revision label.

You can select any type of item by its label. For example, you can select all files with a particular revision label and roll them back to that label, making the revision with that label the tip revision. Then you can compare your working files to the rolled-back revisions.

You can set access rights for labels at the view level or at the folder or item level. You must grant the rights to create labels, edit their properties, and delete them at the view level. However, you can grant the right to move a label (also called *adjust a label*) at the folder or item level.

A label can be frozen, which means no new artifacts can be attached to it, and attached artifacts cannot be detached nor reattached at a different revision. Conversely, non-frozen labels can have all of these modifications. Since many organizations depend on the immutability of frozen labels, a specific security permission is required to freeze or unfreeze a label.

StarTeam actually attaches specific *item revisions* to a label, each of which infers a specific artifact revision. Since each item specifies a parent folder, the artifact is attached in a specific folder. This means that if you move an item, you need to reattach it to any label for which you want to reflect the artifact in its new folder. If you don't reattach a moved item to an existing label, it will continue to be attached to the label in its old folder (which may be what you want).

Even if frozen, both view and revision labels can be cloned. That is, you can create a new label starting out identical to an existing label and then adjust the revisions attached to it. A common practice is to clone a previous label, attach only new file revisions that were created to fix a bug, and use the new label to identify the file revisions for a new build candidate or release candidate.

## Time Stamps and Build Labels

Using a view label as a *time stamp*, you can roll a view back to see everything in the view as it was at the time the label was attached. For example, to see if a particular file was in the beta version of a product, you can roll back the view to the beta label.

You may also use a view label as a *build label*, which allows the QA team to immediately determine what build to test for a fix to any given change request. To use a view label for this purpose:

- It must be designated as a build label.
- It must be created while the **Addressed in build** property for the change request contains the value `Next Build`.

When StarTeam creates the label, each change request with `Next Build` as its **Addressed in build** property will be reset to the build label.

To create a view label, you must select the current configuration of the view. Historical configurations are read only, and adding a label is considered a change. However, if a label already exists for a prior

configuration, you can adjust its name, files and folders can be added to it or detached from it, and so on. You can also move a view label from one revision to another.

For example, suppose your administrator creates a view label before each build, giving that label to the tip revision of each file in the view, and then checks out all the files with that label for the build. If the tip revision of one file does not change for a few weeks (or longer), it can acquire several view labels, while a file that changes frequently may have several revisions with no view labels and other revisions with only one view label.

When you detach a view label from a folder, StarTeam automatically detaches the label from everything in the subtree for which the folder is the root. If you roll back a view to a specific view label and a folder does not have that label, you cannot see the children of that folder and their contents anyway.

You can only create a view label at the view level and only while the configuration is current. However, you can create a view label for the current configuration or for a time in the past. In either of these two cases, StarTeam attaches the new label to the tip revision of each folder, file, change request, task, or topic that belonged to the view at the specified time.

You can also create a view label as a copy of an existing view label or as a copy of the view label currently attached to a promotional state. In these two cases, StarTeam attaches the new label to exactly the same items and revisions as the existing view label.

You can check file revisions out using this label or roll back the view to this label and see all the items associated with that label. For example, if you create the view label *Build 100* as you make Build 100 of your product from a view, all the files in the view will have the label *Build 100*.

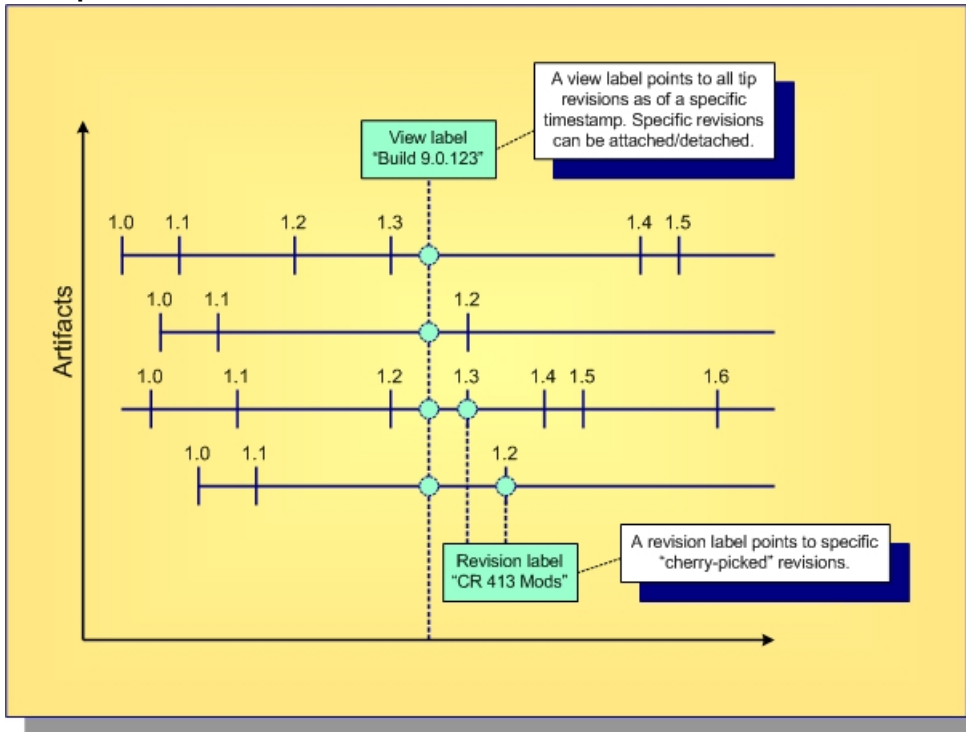
If some items should not be included, you can detach the label from those items individually. For example, if some files should not have that label, select the files then select **Labels > Detach** from the **File** menu or context menu to detach that label. If the files that should not be included all belong to the same folder and are the only files in that folder, use the **Labels** command on the **Folder** menu. For example, if the help files were not checked in until after the view label was attached, you can move that label from the previous revisions of the help files to the newly checked-in help files.


### **Label Access Rights**

You can set access rights that apply to labels at the view level and at the folder/item levels. You set the access rights that allow a user or group to create labels, edit their properties, and delete them at the view level. For example, if you can create a label, you can set its initial properties. However, if you do not have the right to edit label properties, you cannot later freeze or unfreeze that label.

You can attach labels to individual folders or items, detach from them, or move from one of their revisions to another. The access right to move a label is named **Adjust a label**. You can grant or deny these rights at the folder or item level.

## Example View and Revision Labels



 **Note:** Not all items in a view have to be attached to a label. Conversely, an item can be attached to any number of view and revision labels as you like, but only one revision of an item can be attached to any specific label.

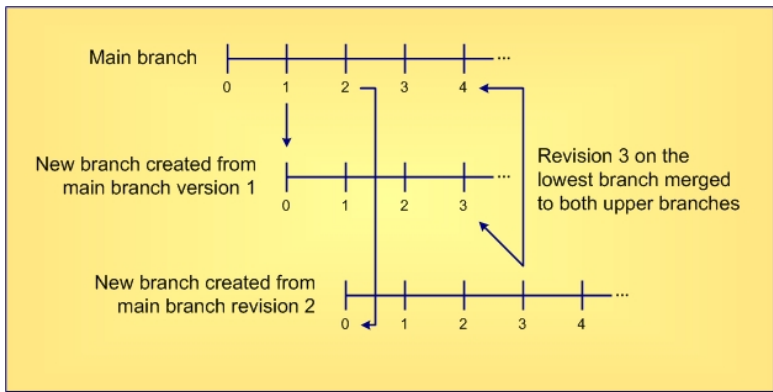
## Branching, Merging and Dot Notation

This topic explores the concepts of branching, merging and dot notation as they relate to StarTeam.

- Branching** Allows an artifact's version stream to be forked. Each fork can be independently modified, receiving its own versions.
- Merging** Propagating a change on one branch to another branch.
- Dot Notation** A dotted decimal notation assigned to artifact revisions to indicate both the branch on which the revision resides and the relative version number of the revision within the branch (for example: 1.4 or 1.2.1.5).

### Branching

There's a big difference between *copying* an artifact and *branching* an artifact. Although copying allows each copy to be modified independently, StarTeam does not know that copied artifacts are related to each other in the repository. In contrast, with branching, StarTeam retains special knowledge about an artifact's branches. This information supports things such as intelligent revision comparison and *three-way merging*, which is discussed below.



When a new artifact is added to the repository, a *main branch* is started and new revisions are added to it. At any time, a parallel *child* branch can be started. In the example above, one child branch is created from main branch version 1, and another child branch is created from main branch version 2. Within a branch, the version number starts over (at zero in this example). New revisions are applied to a specific branch, incrementing the version number on that branch but not affecting other branches. Branching is needed to support parallel development on files. However, there are advantages to allowing non-file artifacts to branch as well. For example, if a defect artifact can be branched, the two branches can be used to track fixes to the same defect that exist in different releases.

## Merging

Inevitably, a change on one branch will need to be propagated to another branch. In the diagram above, version 3 of the artifact on the lowest branch is applied to both of the parent branches. However, you can't just copy a revision from one branch to another branch—this could wipe out changes that are specific to the target branch. Instead, what you want to do is merge the changes from the source to target branch.



**Note:** *Overwriting* the target artifact with the source revision instead of merging it is sometimes the desired effect, for example with binary files.

More specifically, StarTeam stores synchronization information that allows *three-way merging*. The three parts of a merge operation are:

1. The *source revision* containing changes to be propagated.
2. The *target revision* that will be modified.
3. The last source revision common to the source and target branches, known as the *common ancestor*.

For files, merging is done by passing these three file revisions to the **StarTeamFile Compare/Merge** tool. The tool compares both the source revision and target revision to the common ancestor revision and determines two important things:

1. What changes appear in the source revision only that should be propagated to the target revision.
2. What changes appear in the target that may conflict with changes in the source revision.

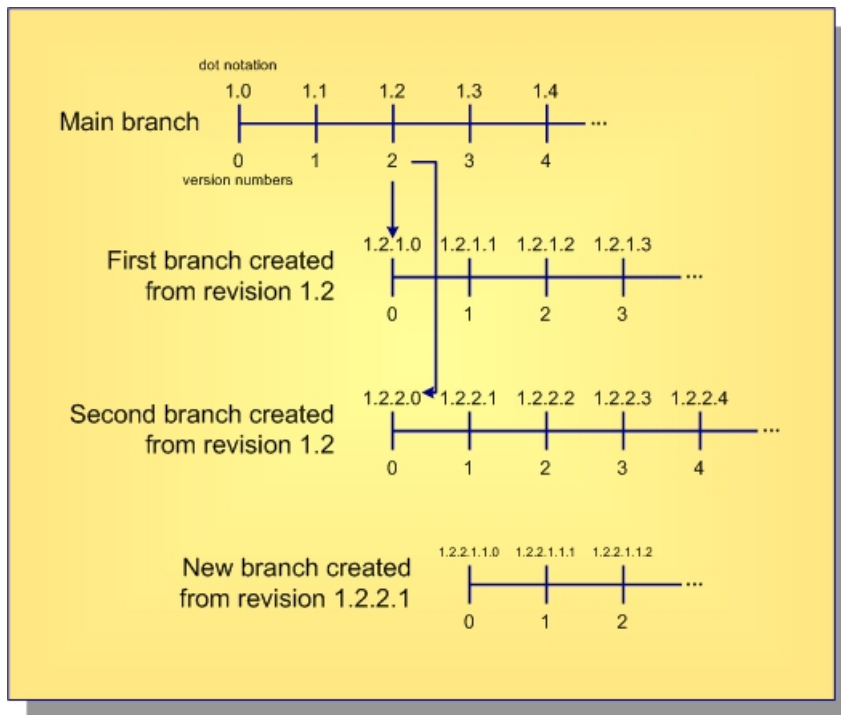
In many cases, merging detects no conflicts, so the **StarTeamFile Compare/Merge** tool automatically propagates the source changes to the target revision. When conflicts are detected (and sometimes even when none are), the **StarTeamFile Compare/Merge** tool displays the differences to a user who can review the differences, resolve conflicts, and approve the final result. The **StarTeamFile Compare/Merge** tool then creates a result file reflecting the target file updated with changes. StarTeam adds the result file to the target revision's branch, creating a new revision.

In the diagram above, revision 3 on the lower child branch was merged to both of the upper branches. When it was merged to the main branch revision 3, it created main branch revision 4, which contains the merge results. (The arrow points to the revision that was created as a result of the merge.) For this merge, the common ancestor between the two branches is main branch revision 2: it is the most recent revision that both branches had in common. When the lower child revision 3 is merged with upper child branch revision 2, upper child branch revision 3 was created. For this merge, the main branch revision 1 is the common ancestor.


For files, there is more to merge than contents: files have other properties such as `name` and `description`. In order to propagate a name change, for example, merging a file requires merging these properties as well. Non-file artifacts that branch also require merging in order to propagate changes.

## Dot Notation

In addition to a version number, StarTeam assigns each revision a dotted-decimal value called a *dot notation*. Whereas the version number is unique within a revision branch, the dot notation value is unique within the entire *revision tree*. An example is shown below.



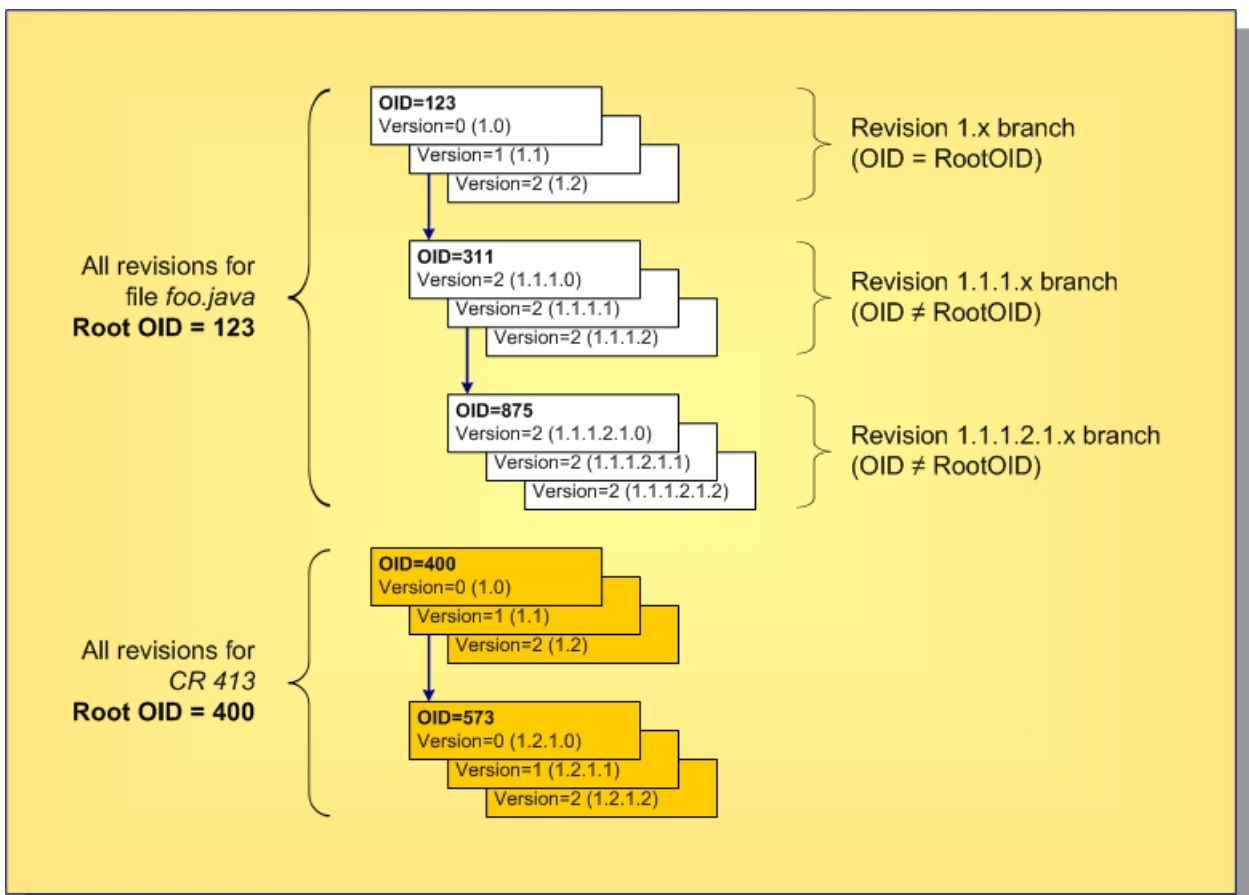
As shown, revisions on the artifact's *main branch* use the single dot notation pair  $1.n$ , where  $1$  indicates that it is the initial (first) branch and  $n$  is the same as the version number. When the artifact is branched from the main branch, revisions on the child branch use the dot notation  $1.m.1.n$ , where  $m$  is the main branch version number from which the branch was created and  $n$  is the version number on the new branch.

 **Note:** An artifact can branch more than once from same point: in the example above, branches  $1.2.1.n$  and  $1.2.2.n$  were both created from main branch revision  $1.2$ . The second  $2$  in  $1.2.2.n$  tells you that this is the second branch from revision  $1.2$ . Branch  $1.2.2.1.1.n$  has three pairs of numbers, telling you that it is a third-level branch, created from parent revision  $1.2.2.1$ .

Artifacts that can't branch (tasks, topics, and requirements) are always on the main branch, so their dot notation is always  $1.n$ .

## Object IDs and Root Object IDs

All revisions in the same revision tree have the same *root object ID* (root OID). All revisions that belong to the same branch have the same object ID (OID). Furthermore, for all revisions on the main branch, the object ID and root object ID are the same. This is illustrated below.



In this example, the file `foo.java` started with OID and root OID 123, and the corresponding 1.n branch has revisions up to 1.2. At revision 1.1, it branched to form the 1.1.1.n branch, which uses the new OID 311. Revision 1.1.1.2 was branched to form the 1.1.1.2.1.n branch with OID 875. But all revisions in the entire branch tree have the original root OID 123. Each revision holds the properties specific to it: name, description, contents, etc.

Also shown is a change request (CR 413) that began with OID and root OID equal to 400. At revision 1.2, it branched to form branch 1.2.1.n with OID 573.

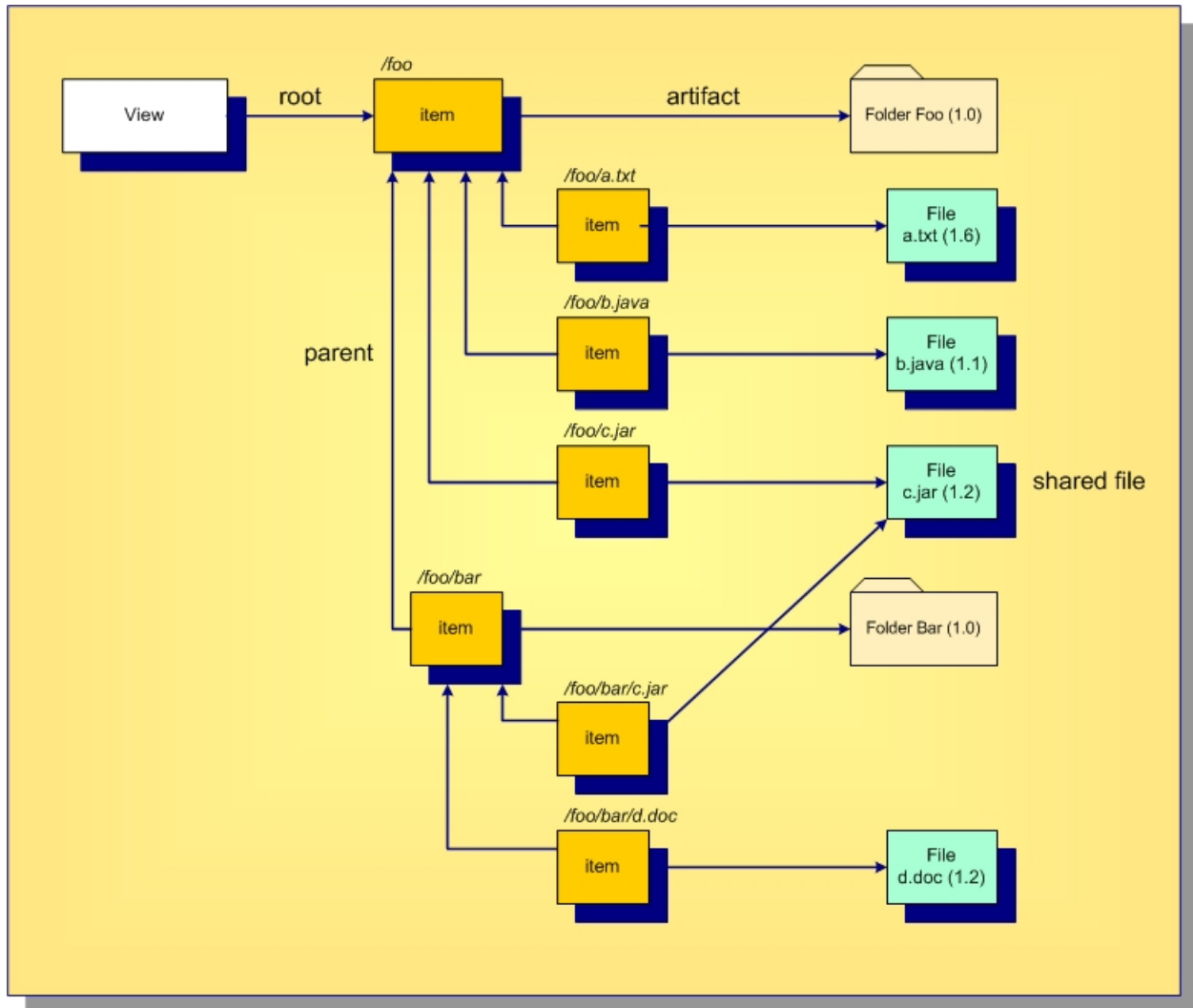
Now, consider a *folder* artifact. Each revision holds properties such as name, description, and `exclude spec` (file patterns to ignore within working folders). What's really different about StarTeam folder artifacts is that they do not have a property that represents their contents.

## Sharing and Cheap Copies

Over time, you will have a lot of artifacts, especially files, and some files will have a lot of branches. Consider the effect of containers: if you have a lot of teams, software components, and releases, you will need a lot of independent projects, subprojects, or other containers to support parallel development and separate maintenance. Often the same files will be needed in each of these containers. How do you get the files you need to each of these containers? Forcing every file to branch in order to get a unique branch in every possible container could be a lot of branching, which is expensive.

StarTeam systems addresses this problem with a technique known as *cheap copies*. This involves creating references to files in a new container. Similar to UNIX links, this happens without actually copying the files themselves (that is, their content or their history). Unlike UNIX links, however, the first time a file is modified via a new reference, it is branched. For this reason, cheap copies are also referred to as “copy on write” sharing. Cheap copies support efficient branching with large projects.

In StarTeam, the folder hierarchy and the contents of each folder are specific to each view. Artifacts can belong to (or more properly be exposed through) any number of views and projects. Items are objects that select specific artifacts, connect them to a specific view, and organize them into a hierarchy. The diagram below shows how this works.



Every view has a *root item*, which always points to a folder artifact. In this example, the root folder name is `foo`. We can make any artifact in the repository belong to this folder by creating an item that points to the artifact we want and the root item as the *parent*. In this example, the files `a.txt`, `b.java`, and `c.jar` and the folder `bar` are all child elements of the root folder `foo`. As you can see, the concept of *path name* is formed by concatenating the names referenced by the item structure. In this view, there is a file whose path name is `/foo/bar/d.doc` because we can get to this artifact via the item path: folder `foo` to folder `bar` to file `d.doc`. If we want to change the folder in which `d.doc` appears, we change the *parent* of its associated item, the artifact itself is not modified. Notice that two items reference the file `c.jar`. This means that this file is contained in two different folders. We say that the file is *shared* in two places. This is analogous to UNIX links that reference the same file, causing it to appear in multiple directories. Sharing allows any artifact to be shared in multiple places. Since artifacts are “heavy” (they contain all the properties) and items are “light”, this is how “cheap copies” are made: we just create items pointing to existing artifacts.

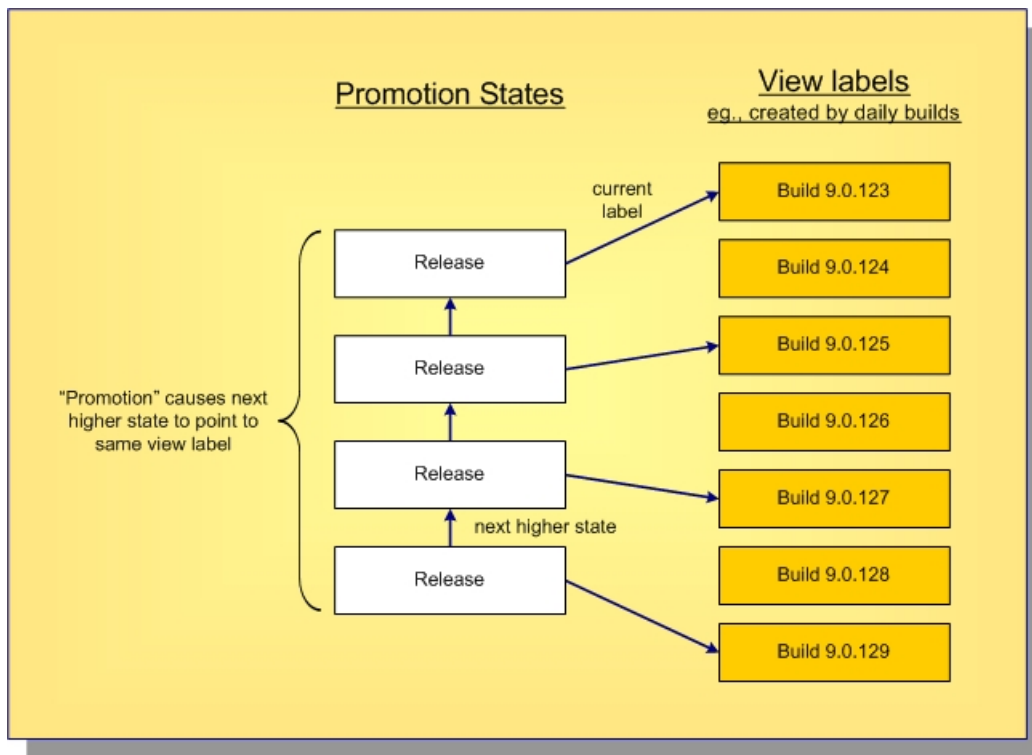
# Promotion States

Promotion states provide an *intra view* change management facility. Promotion states are built on top of view labels, providing an ordered set of states through which items can be promoted within a view. Promotion states are generally used to move the entire view (or most items within it) through a series of steps based on passage of specific verification tests.

After defining a set of promotion states and what view label each state is initially mapped to, you then periodically create new view labels to represent specific view states (such as daily build candidates). Depending on your process, you then typically map the lowest-level state to the new view label and launch the first verification test. After the tests for that state complete, you “promote” it, causing the next higher state to be mapped to that view label. (Multiple states often point to the same view label.) When the tests for the final or “top most” promotion state passes, the view is ready for release, deployment, or whatever your process calls for.

Promotion states allow you to create build scripts, unit test scripts, deployment scripts, and so forth that operate on a specific promotion state without having to be modified to know about new view label names.

An example set of promotion states is shown below.



## Audit Log

The audit log is a record of events that happen to your assets. Open the Audit view to display audit log entries for the selected view.

### Audit Log Events

Events are actions performed on an owner. For example, a file can be checked in or removed from version control. Such events are recorded in the audit log. Most items can be:

- Added
- Branched



- Comment Edited
- Created
- Deleted
- Locked
- Lock Broken
- Modified
- Moved From
- Moved to
- Shared
- Unlocked
- Converted
- Edited
- Item Overwritten (as foreign archive files become native files)
- Vault
- Created
- Modified
- Deleted
- Frozen
- Unfrozen
- Attached
- Moved
- Detached
- Modified

## Audit Fields

This section lists all the audit fields in alphabetical order.

<b>Class Name 1</b>	<p>Values: text</p> <p>Internal Identifier: <code>Class Name 1</code> (contains spaces)</p> <p>The name of the class of items, such as Label, Promotion State, Folder, File, Change Request, Topic, Task, or Trace.</p>
<b>Class Name 2</b>	<p>Values: text</p> <p>Internal Identifier: <code>Class Name 2</code> (contains spaces)</p> <p>The name of the class of items, such as Folder, File, Change Request, Label, Topic, Task, or Trace.</p>
<b>Class Name 3</b>	<p>Values: text</p> <p>Internal Identifier: <code>Class Name 3</code> (contains spaces)</p> <p>The name of the class of items, such as Folder, File, Change Request, Label, Topic, Task, or Trace.</p>
<b>Created By</b>	<p>Values: list of users, &lt;None&gt;</p> <p>Internal Identifier: <code>CreatedUserID</code></p> <p>Always empty because the audit entry is created by the system.</p>
<b>Created Time</b>	<p>Values: date/time</p> <p>Internal Identifier: <code>CreatedTime</code></p>

	The time at which this entry was created.
<b>Deleted By</b>	<p>Values: list of users, &lt;None&gt;</p> <p>Internal Identifier: DeletedUserID</p> <p>The name of the user who deleted an audit entry. Because deleted entries do not appear in the list, this information is unavailable to users.</p>
<b>Deleted Time</b>	<p>Values: date/time</p> <p>Internal Identifier: DeletedTime</p> <p>The time at which an audit entry was deleted. Because deleted entries do not appear in the list, this information is unavailable to users.</p>
<b>Event</b>	<p>Values: Added, Branched, Comment Edited, Created, Deleted, Edited, Item Overwritten, Label Attached, Label Created, Label Deleted, Label Detached, Label Frozen, Label Modified, Label Moved, Label Unfrozen, Lock Broken, Locked, Modified, Moved From, Moved To, Promotion Model Modified, Promotion State Modified, Shared, Unlocked, Vault Converted</p> <p>Internal Identifier: EventID</p> <p>The name of the operation being recorded.</p>
<b>Folder</b>	<p>Values: text</p> <p>Internal Identifier: Folder</p> <p>The name of the folder that stores the audit entry.</p>
<b>Folder Path</b>	<p>Values: text</p> <p>Internal Identifier: Folder Path (contains spaces)</p> <p>The path to the folder that stores the audit entry.</p>
<b>Folder VMID (Advanced)</b>	<p>Values: number</p> <p>Internal Identifier: FolderVMID</p> <p>The ID assigned to the folder that stores the item.</p>
<b>Item 1</b>	<p>Values: text</p> <p>Internal Identifier: Item 1 (contains spaces)</p> <p>Indicates what class 1 item received the audited operation. This can be the name of a file or task, the number of a change request or requirement, or the title of a topic.</p>
<b>Item 1 Info</b>	<p>Values: text</p> <p>Internal Identifier: Info</p> <p>Provides the revision number in dot notation for the class 1 item, if it is revisionable. For example, a label can be a class 1 item and it does not have revisions.</p>
<b>Item 2</b>	<p>Values: text</p> <p>Internal Identifier: Item 2 (contains spaces)</p> <p>Indicates what class 2 item received the audited operation. For example, if a label was attached to a file, the class 1 item is the label and the class 2 item is the file.</p>

<b>Item 2 Info</b>	<p>Values: text</p> <p>Internal Identifier: <code>Info2</code></p> <p>Provides the revision number in dot notation for the class 2 item, if it is revisionable. For example, a label can be a class 2 item and it does not have revisions.</p>
<b>Item 3</b>	<p>Values: text</p> <p>Internal Identifier: <code>Item 3</code> (contains spaces)</p> <p>Indicates what class 3 item received the audited operation. For example, if a label was moved from one revision to a file to another, the class 1 item is the label, the class 2 item is the revision of the file that was initially , and the class 3 item is the final revision of the file.</p>
<b>Item 3 Info</b>	<p>Values: text</p> <p>Internal Identifier: <code>Info3</code></p> <p>Provides the revision number in dot notation for the class 3 item, if it is revisionable. For example, a label can be a class 1 item and it does not have revisions.</p>
<b>Modified By</b>	<p>Values: list of users, &lt;None&gt;</p> <p>Internal Identifier: <code>ModifiedUserID</code></p> <p>Does not apply to audit entries.</p>
<b>Modified Time</b>	<p>Values: date/time</p> <p>Internal Identifier: <code>ModifiedTime</code></p> <p>Does not apply to audit entries.</p>
<b>Object ID</b>	<p>Values: number</p> <p>Internal Identifier: <code>ID</code></p> <p>Each audit entry is assigned an object ID when it is added to a view.</p>
<b>Project</b>	<p>Values: list of projects in this server configuration, &lt;None&gt;</p> <p>Internal Identifier: <code>ProjectID</code></p> <p>The name of the project in which an audit entry was recorded.</p>
<b>Target 1 Class ID (Advanced)</b>	<p>Values: number</p> <p>Internal Identifier: <code>Target 1 Class ID</code> (contains spaces)</p> <p>The ID number assigned to class 1 items or a -1 if there is no ID.</p>
<b>Target 1 Object ID (Advanced)</b>	<p>Values: number</p> <p>Internal Identifier: <code>Target 1 Object ID</code> (contains spaces)</p> <p>The object ID for the class 1 item that received the audited operation or a -1 if there is no ID.</p>
<b>Target 1 Revision Time</b>	<p>Values: date/time</p> <p>Internal Identifier: <code>Target 1 Revision Time</code> (contains spaces)</p> <p>The time at which the last revision was made to the class 1 item that received the audit operation.</p>

<b>Target 2 Class ID (Advanced)</b>	<p>Values: number</p> <p>Internal Identifier: Target 2 Class ID (contains spaces)</p> <p>The ID number assigned to class 2 items or a -1 if there is no ID.</p>
<b>Target 2 Object ID (Advanced)</b>	<p>Values: number</p> <p>Internal Identifier: Target 2 Object ID (contains spaces)</p> <p>The object ID for the class 2 item that received the audited operation or a -1 if there is no ID.</p>
<b>Target 2 Revision Time</b>	<p>Values: number</p> <p>Internal Identifier: Target 2 Revision Time (contains spaces)</p> <p>The time at which the last revision was made to the class 2 item that received the audit operation.</p>
<b>Target 3 Class ID (Advanced)</b>	<p>Values: number</p> <p>Internal Identifier: Target 3 Class ID (contains spaces)</p> <p>The ID number assigned to class 3 items or a -1 if there is no ID.</p>
<b>Target 3 Object ID (Advanced)</b>	<p>Values: number</p> <p>Internal Identifier: Target 3 Object ID (contains spaces)</p> <p>The object ID for the class 3 item that received the audited operation or a -1 if there is no ID.</p>
<b>Target 3 Revision Time</b>	<p>Values: date/time</p> <p>Internal Identifier: Target 3 Revision Time (contains spaces)</p> <p>The time at which the last revision was made to the class 3 item that received the audit operation.</p>
<b>Transaction ID (Advanced)</b>	<p>Values: number</p> <p>Internal Identifier: TransactionID</p> <p>Uniquely identifies the database transaction that contained the update represented by the audit record. (A database transaction can contain multiple updates.) Note that audit records created before the database was upgraded to a StarTeam release that records a <b>Transaction ID</b> will have a <b>Transaction ID</b> of -1.</p>
<b>User</b>	<p>Values: list of users, &lt;None&gt;</p> <p>Internal Identifier: UserID</p> <p>The name of the user who performed the recorded operation.</p>
<b>View</b>	<p>Values: list of views, &lt;None&gt;</p> <p>Internal Identifier: ViewID</p> <p>The name of the view in which an audit entry was recorded.</p>

## Table of Common Operations

The following table lists the generic operations that can be performed with each component.

Operation	File	Change Request	Requirement	Task	Topic	Folder	Audit
Moving	Yes	Yes	No	Yes, except when using Microsoft Project Integration.	Yes	Yes	No
Drag an item to a new location	No	No	No	No	No	Moving a folder moves its contents, child folders, and their contents.	No
Creating shortcuts to items	Yes	Yes	Yes	Yes	Yes	Yes	No
Copying items to a third-party application via a URL	Yes	Yes	Yes	Yes	Yes	Yes	No
Sharing Ctrl+drag item to a new location	Yes	Yes	Yes	Yes	Yes	Yes	No
Branching behavior	Yes	Yes	No	No	No	Yes	No
Configuring to or freezing at a point in the past	Yes	Yes	Yes	Yes	Yes	Yes	No
Locking	Yes	Yes	Yes	Yes	Yes	No	No
Comparing properties of two items of the same type	Yes	Yes	Yes	Yes	Yes	No	No
Comparing properties of two revisions	Yes	Yes	Yes	Yes	Yes	Yes	No
Review revision history	Yes	Yes	Yes	Yes	Yes	Yes	No
Viewing revision properties	Yes	Yes	Yes	Yes	Yes	Yes	No
Editing revision comments	Yes	Yes	Yes	Yes	Yes	Yes	No
Merging revisions	Yes, using Visual Merge.	No, except as a part of merging views.	No, except as a part of merging views.	No, except as a part of merging views.	No, except as a part of merging views.	No, except as a part of merging views, which is often	No

Operation	File	Change Request	Requirement	Task	Topic	Folder	Audit
						done by an administrator	
Finding based on field content	Yes	Yes	Yes	Yes	Yes	No	Yes
Selecting by query	Yes	Yes	Yes	Yes	Yes	No	Yes
Selecting by label	Yes	Yes	Yes	Yes	Yes	No	No
Label revisions	Yes	Yes	Yes	Yes	Yes	Yes	No
Viewing references	Yes	Yes	Yes	Yes	Yes	Yes	No
Linking to folders and items	Yes	Yes	Yes	Yes	Yes	Yes	No
Printing a default report for selected items	Yes	Yes	Yes	Yes	Yes	No	No
Sending items as email	No	Yes	Yes	Yes	Yes	No	Yes
Receiving email notification about changes (when notification is enabled by administrator )	No	Yes. Changes in responsibility only.	Yes. All changes in items for which you are responsible.	Yes. All changes in items for which you are responsible.	Yes. All changes in items for which you are responsible.	No	No
Controlling system tray notification	No	Yes	Yes	Yes	Yes	No	No
Marking items as read/unread	No	Yes. You can also mark trees as read/unread.	Yes. You can also mark trees as read/unread.	Yes. You can also mark trees as read/unread.	Yes. You can also mark trees as read/unread.	No	No
Flagging items	Yes	Yes	Yes	Yes	Yes	No	No
Deleting	Yes	Yes	Yes	Yes	Yes	Yes	No
Setting access rights. Normally performed by	Yes	Yes	Yes	Yes	Yes	Yes	No


Operation	File	Change Request	Requirement	Task	Topic	Folder	Audit
Administrator							
Creating reports	Yes	Yes	Yes	Yes	Yes	No	Yes
Creating charts	Yes	Yes	Yes	Yes	Yes	No	Yes

## User-Defined Property Fields

StarTeam Enterprise Advantage and Enterprise licenses enable you to customize the repository by adding any number of user-defined property fields to files, change requests, requirements, topics, and tasks.

You can also change the properties of some existing items. For example, you can customize the **Priority** change request field. Instead of using the default values (**Yes** and **No**), your company can change the item properties so that StarTeam prioritizes change requests on a scale from 1 (high) to 10 (low).

The application adds new property fields and modified property fields to the database used by your current server configuration. These fields are available in the same locations as other item property fields.

 **Note:** When you customize the database, you should use the `stcmd` command line to exclusively lock the StarTeam Server.

## Displaying Custom Property Fields


1. Select the folder in the Server Explorer or the Eclipse Explorers that contains the task that you want to modify.
2. Do one of the following:
  - Open an item view, such as, the **Change Request**, **Requirement**, **Task**, or **Topic** views.
  - Select a file in either the Server Explorer or in one of the Eclipse Explorers.
3. Choose **Properties**. The **Properties** dialog box opens.
4. Do one of the following:
  - If selecting a file from one of the Eclipse Explorers, expand **StarTeam Provider**, select the **File** node, and click the **Custom** tab.
  - If working in one of the item views, click the **Custom** tab.

The **Custom** tab contains any custom fields created for the selected item.

5. Click **OK**.


## Creating Content Fields


Content property type fields can be used to store rich text values, such as HTML MIME types and the StarTeam Cross-Platform Client provides a rich text editor for users to edit them. Use the **Text** property type field to store standard text with no markup.

 **Note:** A default value cannot be set for a this field because they normally contain unique data.

1. Select **[Component] > Advanced > Customize**. **[Component]** corresponds to the component for which you are creating a property field. The **Customize** dialog box opens.

2. Review the **Customize** dialog box to find the fields that can be customized. The fields and icon graphics accompanying them are as follows:
  - A field with a pencil in the center of the icon is an application field. It is always an enumerated type and is fully customizable. You can add, disable, rename, and reorder its values.
  - A new field icon displays the head and shoulders of a person. It can be one of several types and is fully customizable. If the new field is disabled, the icon is greyed-out.
  - An application field with an icon containing a small yellow lock in the lower left corner is a restricted enumerated type. You can change only the names the application displays for the values of this field.
3. Click **Add**. The **Add Field** dialog box opens.
4. Type the name for the new field to be used by the database in the **Field name** field. Use only alphanumeric characters and no spaces in this name. The name should be less than 31 ASCII characters (including `usr_`) and not contain the following characters, which are not accepted by one or more of the databases that the application supports: `= \ \ . ^ $ @ , ; ! : # * & < > ? - / % | [ ] ( ) ) + "`


 **Note:** Be careful about selecting the field name, as it cannot be changed once you click **OK**.
5. Type the name that the application will display to users in the **Display Name** field.
6. Select **Content** from the **Type** list.
 

 **Note:** The default maximum length is 255 characters and cannot be changed.
7. Click **OK**.

## Creating Date and Time Fields

There are two date and time property types that can be used to create new date or date and time fields. The **Date** type has only a date component, while the **Date/Time** type has both a date and a time component.

1. Select **[Component] > Advanced > Customize**. **[Component]** corresponds to the component for which you are creating a property field. The **Customize** dialog box opens.
2. Review the **Customize** dialog box to find the fields that can be customized. The fields and icon graphics accompanying them are as follows:
  - A field with a pencil in the center of the icon is an application field. It is always an enumerated type and is fully customizable. You can add, disable, rename, and reorder its values.
  - A new field icon displays the head and shoulders of a person. It can be one of several types and is fully customizable. If the new field is disabled, the icon is greyed-out.
  - An application field with an icon containing a small yellow lock in the lower left corner is a restricted enumerated type. You can change only the names the application displays for the values of this field.
3. Click **Add**. The **Add Field** dialog box opens.
4. Type the name for the new field to be used by the database in the **Field name** field. Use only alphanumeric characters and no spaces in this name. The name should be less than 31 ASCII characters (including `usr_`) and not contain the following characters, which are not accepted by one or more of the databases that the application supports: `= \ \ . ^ $ @ , ; ! : # * & < > ? - / % | [ ] ( ) ) + "`

 **Note:** Be careful about selecting the field name, as it cannot be changed once you click **OK**.
5. Type the name that the application will display to users in the **Display Name** field.
6. Select **Date** or **Date/Time** from the **Type** list. The **Date** type stores date values without requiring inclusion of **Time**.
 

Optionally, select the **Default value** check box and type in a default date or time, using the appropriate format for your locale. This value automatically becomes the value used for all existing items that have the field as a property. It also becomes the default value for newly created items. You can change this value manually on the **Custom** tab of the item **Property** dialog box.





**Note:** If you do not set a default, no value is placed in the database for any item with this field, unless you set the field to a value manually.

7. Click **OK**.

## Creating Enumerated Fields

The order in which enumerated values appear in list boxes in the **Query** and other dialog boxes is the order in which they appear in the **Add Field** or **Modify Field** dialog box. This can be a code order or even alphabetic order, but only if they appear in that way in the dialog box. You can use drag-and-drop to rearrange the values.

1. Select **[Component] > Advanced > Customize**. **[Component]** corresponds to the component for which you are creating a property field. The **Customize** dialog box opens.
2. Review the **Customize** dialog box to find the fields that can be customized. The fields and icon graphics accompanying them are as follows:
  - A field with a pencil in the center of the icon is an application field. It is always an enumerated type and is fully customizable. You can add, disable, rename, and reorder its values.
  - A new field icon displays the head and shoulders of a person. It can be one of several types and is fully customizable. If the new field is disabled, the icon is greyed-out.
  - An application field with an icon containing a small yellow lock in the lower left corner is a restricted enumerated type. You can change only the names the application displays for the values of this field.
3. Click **Add**. The **Add Field** dialog box opens.
4. Type the name for the new field to be used by the database in the **Field name** field. Use only alphanumeric characters and no spaces in this name. The name should be less than 31 ASCII characters (including `usr_`) and not contain the following characters, which are not accepted by one or more of the databases that the application supports: `= \\. ^$@, ; ! : # * & < > ? - / % | [ ] ( ) + "`



**Note:** Be careful about selecting the field name, as it cannot be changed once you click **OK**.

5. Type the name that the application will display to users in the **Display Name** field.
6. Select **Enumerated** from the **Type** list.
7. Click **Add** to enter the first value. The **Add Value** dialog box opens. StarTeam reserves the numeric codes from 0 to 100, so this dialog shows numbers starting with 101.
8. Use the displayed code or type another in the **Code** text box.
9. Type the name for this enumerated value in the **Name** text box and click **OK**.
10. Use drag-and-drop to arrange the values in the **Possible Values** list box.
11. Select **Supports multi-select** to provide users with the ability to select more than one value.
12. In the **Default Value** box select one or multiple enumerated values as the default for users to select. This value automatically becomes the value used for all existing items that have the field as a property. It also becomes the default value for newly created items. You can change this value manually on the **Custom** tab of the item **Property** dialog box.

Many users create a value of **<none>** to use as the default, to indicate that no value is really being set. They also place this value first in the enumerated list, so that it displays at the top or bottom of the upper pane when the column for the field is sorted. If you are entering a date or time, use the appropriate format for your locale.
13. *Optionally:* After you have added the values, you can arrange them hierarchically and sort the values in parent/child relationships as desired.
  - a) In the **Possible Values** list, select a code and click **Edit**.

If you are adding a new value, click **Add**.
  - b) Select **Make child of** and choose a value in the list to arrange the values in parent/child arrangements.



**Tip:** Alternatively, you can also drag and drop the values in the list to arrange them hierarchically. When dragging a node, it becomes a sibling of the node it is dropped onto, and it is placed immediately after that node.

c) Click **OK**.

14. Verify that the codes you have selected are the ones that you want. They cannot be changed once you exit the dialog box and click **OK**.

## Creating Group and Group List Fields

Group fields can be used to store the user group values defined on the server. There are two group property types that can be used to create new group fields. The **Group** type allows users to only select one group, while the **Group List** is a multi-select option that allows multiple group values to be selected.



**Note:** A default value cannot be established for a group field, because if the **Group** is deleted, the value becomes invalid.

1. Select **[Component] > Advanced > Customize**. **[Component]** corresponds to the component for which you are creating a property field. The **Customize** dialog box opens.
2. Review the **Customize** dialog box to find the fields that can be customized. The fields and icon graphics accompanying them are as follows:
  - A field with a pencil in the center of the icon is an application field. It is always an enumerated type and is fully customizable. You can add, disable, rename, and reorder its values.
  - A new field icon displays the head and shoulders of a person. It can be one of several types and is fully customizable. If the new field is disabled, the icon is greyed-out.
  - An application field with an icon containing a small yellow lock in the lower left corner is a restricted enumerated type. You can change only the names the application displays for the values of this field.
3. Click **Add**. The **Add Field** dialog box opens.
4. Type the name for the new field to be used by the database in the **Field name** field. Use only alphanumeric characters and no spaces in this name. The name should be less than 31 ASCII characters (including `usr_`) and not contain the following characters, which are not accepted by one or more of the databases that the application supports: `= \ \ . ^ $ @ , ; ! : # * & < > ? - / / % | [ ] ( ) + "`



**Note:** Be careful about selecting the field name, as it cannot be changed once you click **OK**.

5. Type the name that the application will display to users in the **Display Name** field.
6. Select **Group** (for a single group value selection) or **Group List** (for multiple group values selection) from the **Type** list.
7. Select the **Input required** check box to make this a required field. Required fields are rarely used with files, because they affect the **File Properties** dialog box only.
8. Click **OK**.

## Creating Map Fields

Map fields can be used to store structured values on items where the value types are not required to be the same or exist on every item instance.



**Note:** A default value cannot be set for a this field because they normally contain unique data.

1. Select **[Component] > Advanced > Customize**. **[Component]** corresponds to the component for which you are creating a property field. The **Customize** dialog box opens.
2. Review the **Customize** dialog box to find the fields that can be customized. The fields and icon graphics accompanying them are as follows:

- A field with a pencil in the center of the icon is an application field. It is always an enumerated type and is fully customizable. You can add, disable, rename, and reorder its values.
- A new field icon displays the head and shoulders of a person. It can be one of several types and is fully customizable. If the new field is disabled, the icon is greyed-out.
- An application field with an icon containing a small yellow lock in the lower left corner is a restricted enumerated type. You can change only the names the application displays for the values of this field.

3. Click **Add**. The **Add Field** dialog box opens.

4. Type the name for the new field to be used by the database in the **Field name** field. Use only alphanumeric characters and no spaces in this name. The name should be less than 31 ASCII characters (including `usr_`) and not contain the following characters, which are not accepted by one or more of the databases that the application supports: `= \\.^$@, ; ! : # * & < > ? - / % | [ ] ( ) + "`



**Note:** Be careful about selecting the field name, as it cannot be changed once you click **OK**.

5. Type the name that the application will display to users in the **Display Name** field.

6. Select **Map** from the **Type** list.



**Note:** The default maximum length is 255 characters and cannot be changed.

7. Click **OK**.

## Creating Rich Content Fields

A default value cannot be set for a rich content field because most rich content fields contain unique data.

1. Select **[Component] > Advanced > Customize**. **[Component]** corresponds to the component for which you are creating a property field. The **Customize** dialog box opens.

2. Review the **Customize** dialog box to find the fields that can be customized. The fields and icon graphics accompanying them are as follows:

- A field with a pencil in the center of the icon is an application field. It is always an enumerated type and is fully customizable. You can add, disable, rename, and reorder its values.
- A new field icon displays the head and shoulders of a person. It can be one of several types and is fully customizable. If the new field is disabled, the icon is greyed-out.
- An application field with an icon containing a small yellow lock in the lower left corner is a restricted enumerated type. You can change only the names the application displays for the values of this field.

3. Click **Add**. The **Add Field** dialog box opens.

4. Type the name for the new field to be used by the database in the **Field name** field. Use only alphanumeric characters and no spaces in this name. The name should be less than 31 ASCII characters (including `usr_`) and not contain the following characters, which are not accepted by one or more of the databases that the application supports: `= \\.^$@, ; ! : # * & < > ? - / % | [ ] ( ) + "`



**Note:** Be careful about selecting the field name, as it cannot be changed once you click **OK**.

5. Type the name that the application will display to users in the **Display Name** field.

6. From the **Add Field** dialog box, select **Rich Content** from the **Type** field.


7. Click **OK**.

## Creating Text Fields

Text property type fields store standard text with no markup. Use the Content property type to store rich text values such as HTML.

1. Select **[Component] > Advanced > Customize**. **[Component]** corresponds to the component for which you are creating a property field. The **Customize** dialog box opens.


2. Review the **Customize** dialog box to find the fields that can be customized. The fields and icon graphics accompanying them are as follows:
  - A field with a pencil in the center of the icon is an application field. It is always an enumerated type and is fully customizable. You can add, disable, rename, and reorder its values.
  - A new field icon displays the head and shoulders of a person. It can be one of several types and is fully customizable. If the new field is disabled, the icon is greyed-out.
  - An application field with an icon containing a small yellow lock in the lower left corner is a restricted enumerated type. You can change only the names the application displays for the values of this field.
3. Click **Add**. The **Add Field** dialog box opens.
4. Type the name for the new field to be used by the database in the **Field name** field. Use only alphanumeric characters and no spaces in this name. The name should be less than 31 ASCII characters (including `usr_`) and not contain the following characters, which are not accepted by one or more of the databases that the application supports: `= \\.^$@, ; ! : # * & < > ? - / % | [ ] ( ) + "`

 **Note:** Be careful about selecting the field name, as it cannot be changed once you click **OK**.
5. Type the name that the application will display to users in the **Display Name** field.
6. Select **Text** from the **Type** list.
7. Use the default maximum length (255 characters) or type a number of characters from 2 to 20,000 in the **Length** text box.
8. Verify the **Length** is adequate, because you cannot change it after exiting the dialog.
9. Select the **Input required** check box if you want to make the field required. Required fields are rarely used with files, because they affect the **File Properties** dialog box only.
10. Click **OK**.

## Creating Time Span Fields

Time span property fields can be added to specify a duration of time in days, hours, minutes, and seconds.

1. Select **[Component] > Advanced > Customize**. **[Component]** corresponds to the component for which you are creating a property field. The **Customize** dialog box opens.
2. Review the **Customize** dialog box to find the fields that can be customized. The fields and icon graphics accompanying them are as follows:
  - A field with a pencil in the center of the icon is an application field. It is always an enumerated type and is fully customizable. You can add, disable, rename, and reorder its values.
  - A new field icon displays the head and shoulders of a person. It can be one of several types and is fully customizable. If the new field is disabled, the icon is greyed-out.
  - An application field with an icon containing a small yellow lock in the lower left corner is a restricted enumerated type. You can change only the names the application displays for the values of this field.
3. Click **Add**. The **Add Field** dialog box opens.
4. Type the name for the new field to be used by the database in the **Field name** field. Use only alphanumeric characters and no spaces in this name. The name should be less than 31 ASCII characters (including `usr_`) and not contain the following characters, which are not accepted by one or more of the databases that the application supports: `= \\.^$@, ; ! : # * & < > ? - / % | [ ] ( ) + "`

 **Note:** Be careful about selecting the field name, as it cannot be changed once you click **OK**.
5. Type the name that the application will display to users in the **Display Name** field.
6. Select **Time span** from the **Type** list.

Optionally, select the default value **Time Span Days** check box and type in a default value for the number of days and/or select a default time (H = hours, M = minutes, S = seconds). This value automatically becomes the value used for all existing items that have the field as a property. It also

becomes the default value for newly created items. You can change this value manually on the **Custom** tab of the item **Property** dialog box.



**Note:** If you do not set a default, no value is placed in the database for any item with this field, unless you set the field to a value manually.

7. Click **OK**.

## Creating User and User List Fields

User fields can be used to store the user values defined on the server. There are two user property types that can be used to create new user fields. The **User** type allows for the selection of only one user, while the **User List** is a multi-select option that allows multiple user values to be selected.



**Note:** A default value cannot be established for a this field because if the data is deleted, the value becomes invalid.

1. Select **[Component] > Advanced > Customize**. **[Component]** corresponds to the component for which you are creating a property field. The **Customize** dialog box opens.
2. Review the **Customize** dialog box to find the fields that can be customized. The fields and icon graphics accompanying them are as follows:
  - A field with a pencil in the center of the icon is an application field. It is always an enumerated type and is fully customizable. You can add, disable, rename, and reorder its values.
  - A new field icon displays the head and shoulders of a person. It can be one of several types and is fully customizable. If the new field is disabled, the icon is greyed-out.
  - An application field with an icon containing a small yellow lock in the lower left corner is a restricted enumerated type. You can change only the names the application displays for the values of this field.
3. Click **Add**. The **Add Field** dialog box opens.
4. Type the name for the new field to be used by the database in the **Field name** field. Use only alphanumeric characters and no spaces in this name. The name should be less than 31 ASCII characters (including `usr_`) and not contain the following characters, which are not accepted by one or more of the databases that the application supports: `= \ \ . ^ $ @ , ; ! : # * & < > ? - / % | [ ] ( ) + "`
  - **Note:** Be careful about selecting the field name, as it cannot be changed once you click **OK**.
5. Type the name that the application will display to users in the **Display Name** field.
6. Select **User** (for a single user value selection) or **User List** (for multiple user values selection) from the **Type** list.
7. Select the **Input required** check box to make this a required field. Required fields are rarely used with files, because they affect the **File Properties** dialog box only.
8. Click **OK**.

## Projects

A project is a way to group related items (such as files and change requests) hierarchically. Views and folders enable you to organize these related items more efficiently. For example, if you create a project for a software product, the files containing the product's functional specification, marketing requirements document, source code, and test suites can each be stored in separate folders.

Views can be used in a variety of ways. For example, different views can be used so that developers see only the project's source code folder and its child folders, marketing personnel see only the project's marketing folder and its child folders, and so on. In this case, each view has a different folder as its root. Views also support branching and parallel development

At the view level or item by item, you can branch data such as files and change requests. The branching enables you to create a special variation of your product. For example, you can start on the 2.0 version of your product without hampering the creation of service packs for the 1.0 version

You can create a project on any StarTeam Server configuration, if your computer has access to that server and you have been granted the rights needed to create projects in that location. When you create a project, you must provide a project name and designate a working folder location for the project's root (initial) folder. The initial view of the project is created at the same time you create the project. It has the same name as the project, although you can change the name later if you choose. The root folder is also created at this time. If your computer is not currently set up to access the server on which the project will reside, you can add access to that server as part of creating the project.

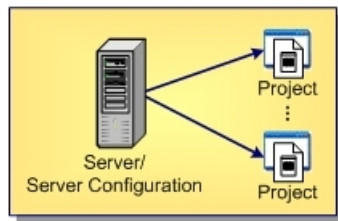
**!** **Important:** The StarTeam Server creates new projects with only the **File** type pre-selected as a default for new views. Users can still change the project properties after the project is created, and they can change the item types included for any given new view. However, if the user changes nothing, by default new views will only include files when they are created. Adding other item types to the **Project Properties** (after the view is created) will NOT populate the items that were contained in the parent view (but left out during New View creation). If the user wants to bring the previous items into the new view, they must retrieve them by Rebasing from the parent view.

## Project Structure

An instance of the StarTeam Server controls the storage of your files. Each StarTeam Server instance runs a server configuration. Below are some diagrams illustrating how all these pieces fit and work together.

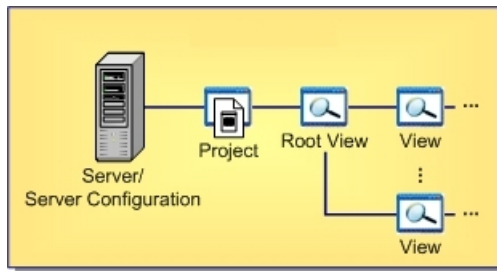
### Server-level Hierarchy

The server can manage any number of projects.



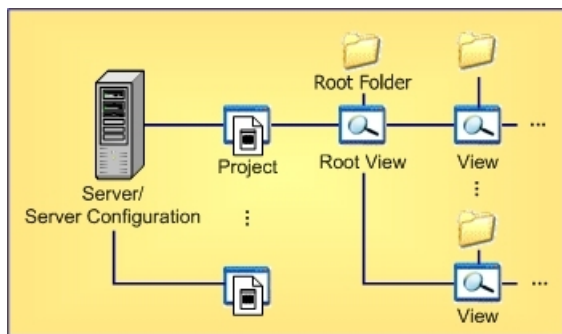
### Project-level Hierarchy

Each project has one root view and any number of child views.



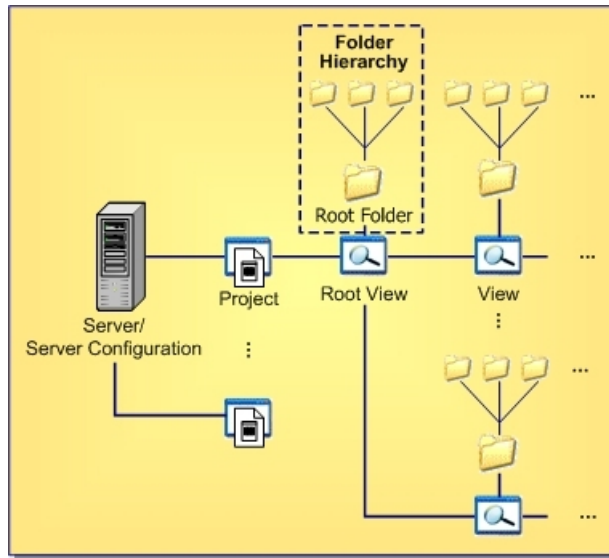
### View-level Hierarchy

The root view and every child view has one application folder as a root folder.



## Folder-level Hierarchy

An application root folder can have any hierarchy of child folders. This is called the folder hierarchy.



## Guidelines for Keeping Projects Autonomous

The time-honored programming tenets of high cohesion and low coupling apply to StarTeam projects as well. The more independent your StarTeam projects are, the easier they are to secure, administer, and even separate from the original StarTeam configuration if necessary. Keeping projects autonomous means keeping cross-project links and shares to a minimum, avoiding them completely if possible.

Below are some guidelines for deciding what should be in the same project:

- A project should be used to manage the life-cycle artifacts of a cohesive application set or application component set. Examples are a commercial software product or a foundation library package. Multiple application or component sets can be managed in a single project if they are interrelated and generally enhanced and released together.
- A project should include all of the artifacts required to manage the life-cycle of the supported applications or components. This includes early life-cycle artifacts such as requirements documents, modeling diagrams, and design documents, as well as construction phase artifacts such as source files, images, and resource files; and later-phase artifacts such as test scripts and applications.
- A project should include all artifacts authored in life-cycle phases as well as non-authored artifacts required to perform authoring. This includes, for example, all files authored by the IDEs such as workspace/project files, source code, and resource files. It also includes “input files” such as .h, .lib, .jar, or .dll files that are authored or generated elsewhere but required by the project’s IDEs or build processes. Input files may originate from third parties or from other projects in the same or other StarTeam configurations. (Transferring artifacts from one project to another is discussed further later).
- Files directly generated from authored files such as .obj, .class, and .lib files generally do not need to be checked into the project. However, it is common practice to check in “final” binaries such as .jar, .war, and .exe files that are delivered to other projects, engineering test, QA, or other deployment phases. The need to place generated files under version control is highly dependent on your own development, testing, and release methodologies.
- A project should have a long-term view of the products or components it supports. That is, it should house all artifacts generated over multiple iterations through the lifecycle. This means that the project supports multiple versions of its applications or components, representing the complete history of those modules.
- StarTeam works best when “work” artifacts (change requests, tasks, topics, and requirements) related to a project’s files are stored in the same project. This allows, for example, a change request to be

entered, tracked, and linked to the files in the same project to which the change request is related. This approach requires some special considerations for activities such as “change request triaging” and cross-project reporting. These issues are discussed later.

Some customers have attempted to use projects to separate development phases (for example, design and development) or development artifact types (like files and change requests). The artifacts are then interrelated by sometimes heavy use of links or shares. However, experience has found that copious use of shares – especially cross-project shares – results in difficulties in version control, reporting, security, and even performance. We suggest that artifacts related to the same applications and components, even though of different types/life-cycle relevance, should be managed in the same project.

### **Scenario 1: A Simple Client/Server Application**

A commercial software application consists of a server written in C++ and a single client, also written in C++. Furthermore, the client and server modules share a fair amount of source code and IDE projects that generate common DLLs. The client and server modules are generally enhanced and released together.

In this scenario, a single StarTeam project should be used to manage the combined files of both the client and server modules. The sharing of source code and shared release schedules suggest that the modules are cohesive parts of a single application. Requirements, design documents, change requests, and other artifacts required for all life-cycle phases of the client and server modules should be managed in the same project.

### **Scenario 2: An Independent Client Module**

A new Java client application is developed that uses the same server described in Example 1. Building and compiling the Java client requires some of the header files and one of the DLLs used by the server to produce a JNI wrapper, but no other source files. Furthermore, the Java application accesses other third-party servers and is generally enhanced and released on independent schedules from those used by the client/server modules.

In this scenario, it is reasonable to use a separate StarTeam project to manage the Java client’s artifacts. The latest header files and generated DLL needed by the Java client are checked into a “external components” folder by the build process used in the client/server project. All change requests, tasks, and other life-cycle objects related to the Java client are managed in the same project.

### **Scenario 3: A Complex Financial Application Suite**

A complex application suite consists of a set of foundation components and nearly 100 separate applications, divided into five functional areas: accounting, insurance, forecasting, etc. The applications are developed by different teams and all use the foundation components, which are jointly maintained by the development teams. Applications within a functional area are highly interrelated, but applications between functional areas are fairly independent. The foundation component library is enhanced and released on its own schedule, but the entire application suite is released as a commercial product in coordinated releases.

Although the entire application suite is interrelated, multiple projects should be used due to the overall application suite size. The foundation components are managed in one project, and each of the five functional areas utilize a separate project to manage the corresponding applications (six projects total). The foundation project is enhanced, built, and then “deployed” to each of the functional area projects by checking in generated jar files. Each development team generally opens only one project to perform their normal work. However, a special build script (using the StarTeam SDK) is used to extract files from multiple projects and generate “whole suite” builds. The build script also automates the management of common view labels and promotion states across projects.

## **Cross-Project Activity Support**

Regardless of how you partition your projects, you may find that certain life-cycle activities span multiple projects. Some examples of these scenarios and how they can be addressed are provided below:



<b>Multi-project Builds</b>	Build procedures that require files from multiple projects can use the StarTeam SDK, which can open multiple projects at the same time. Alternatively, iterative calls to the StarTeam command line tool (stcmd) can be used to check out files from each of the required projects.
<b>Defect Triaging</b>	When a new defect is discovered, it is often entered as a change request before the module that must be repaired is known. This means that, if change requests are normally managed in the project containing the files that will be modified, a paradox exists in determining where to create the change request in the first place. A project leader or other person usually “triages” the change request by assigning it to whoever he or she thinks should handle it. As the change request is analyzed and addressed, the module that is ultimately modified to address it may not be known for awhile. A simple solution for this scenario is to place all change requests in a well known place (perhaps a “new defects” project) and “move” (not copy) them to the appropriate project as needed.
<b>Cross-project Reporting</b>	Currently, StarTeam does not provide built-in, cross-project reports. Consequently, if you want to generate reports such as “all open change requests across all projects” or “cross-server file metrics”, your best option is to use Datamart to harvest and report on change requests from multiple projects and even multiple configurations. Another option is to use the StarTeam SDK to write custom reporting applications.

## How to Handle Cross-Project File Dependencies

When projects are highly cohesive, cross-project dependencies will be minimal, yet cross-project file relationships may still occur. Some files, either authored or generated, may need to be used in multiple projects.

The impulsive way to handle this situation may be to share the co-dependent files from one project to another. On the surface, this approach works, but experience has shown that cross-project sharing is problematic for several reasons:

- If a shared item’s behavior is configured to “floating”, changes from the parent project flow into the child project immediately, sometimes at inconvenient times. Many StarTeam users find that they need to manage the update propagation on a more scheduled basis.
- If a shared item’s behavior is configured to a specific timestamp, it must be occasionally adjusted to allow updates to propagate. This makes the shared item read-only, and continually adjusting the configuration timestamp for a large number of shares can become burdensome.
- If a shared item’s branch-on-change property is set to “true” (perhaps accidentally), and a change is made to the child share, the object will branch in the child project. This severs *automatic promotion* of changes from the parent item to the child share. If the child share is a folder, an innocuous change such as modifying the working folder will cause the folder object to branch
- When an update is made to an item, the entire share tree must be locked. As the share tree grows, update performance is impacted proportionately.
- Normally, when obsolete views and projects are deleted, the StarTeam purge utility can be used to return unused database records and archive files. However, shares used by a deleted project or view can prevent the purge utility from achieving the expected reduction in database and archive size. In short, it might not be possible to reduce the size of servers that use cross-product sharing.

Because of these reasons, other techniques have proven to be more effective at handling cross-project file dependencies. Below are some alternatives to sharing that work in different situations. In the end, shares still may be the most appropriate way to expose files from one project to another, but the approaches below should be considered first.

### Deployment Approach

If a project “owns” a set of files that must be checked into other projects, you can establish a process in which the files are periodically “deployed”. This means that the file set is checked into the target project(s) on an as-needed basis, perhaps by a script. Often, a build script is a good place to embed the deployment

task, especially if the file(s) to be deployed are generated by the build. Keep in mind that checking in the same file multiple times does not (generally) increase the size of the vault, each unique file revision is only stored once.

### Configuration Approach

Sometimes the co-dependent files don't need to be checked into each project, but they need to participate in a common process such as a build or delivery process. In these cases, a simple configuration file (for example, XML) can be constructed that defines the files that participate in the process. If the file is checked into an established location and updated when the configuration changes, then build, release, or other scripts can check out the configuration file, parse it, and base their processing on its instructions.

### Link Approach

In lieu of shares, links can be used to connect objects across servers. Links do not possess many of the problems exhibited by shares, and they can be pinned, unpinned, and moved to refer to different object revisions. The downside of using links is that they are un-typed and possess no properties other than a link comment to identify their purpose.

## Working with Projects

This section contains topics showing you how to work with projects.

### Sharing Workspace Projects

Follow the high-level steps below when using the **Share Project** wizard.

These steps assume that:

- You have not yet created a connection with a StarTeam Server configuration.
- You have already created multiple Eclipse projects in the Workspace to share.
- You have pre-existing StarTeam projects created for the StarTeam Server configuration that you are sharing with the Eclipse projects.

1. Open the **Share Project** wizard.
2. If necessary, create a new StarTeam connection.
3. Log on to a StarTeam Server.
4. Enter repository location information.
5. Synchronize your workspace project with the StarTeam repository.
6. Check in the workspace project files to the repository.

### Opening the Share Project Wizard

1. In the **Navigator** or **Package Explorer**, select the root nodes of the projects you want to share.
2. Right click on the selected projects and choose **Team > Share Project**. The **Share Project** wizard opens.
3. Select StarTeam as the repository type and click **Next**. The next screen of the wizard opens enabling you to configure your workspace project to use the StarTeam repository.

### Creating a New Connection

1. Select **Create a new StarTeam** connection and click **Next**.
2. Type a unique, easy-to-remember description in the **Server description** text box.  
The server description is not case-sensitive and can contain colons ( : ).
3. Type the computer name or IP address in the **Server address** text box.  
See your administrator for the server address, protocol, and endpoint information.

4. Type the endpoint (TCP/IP port number) associated with the protocol in the **TCP/IP endpoint** text box.
5. Click **Test Connection**.
6. Choose to specify any of the following optional settings:
  - Check the **Save Password** option to enable the **User Name** and **Password** text boxes.
  - Select an **Encryption** type to encrypt data transferred between your workstation and the server. Encryption protects files and other project information from being read by unauthorized parties over unsecured network lines. The encryption types are ordered (top to bottom) based on speed. Each type is slower, but safer, than the type that precedes it.
  - Check the option to **Compress transferred** data if you want to use compression.
7. Click **Next** when you are finished.

### Logging on Server with Existing Connection

1. Select **Use an existing StarTeam connection**, and choose your server configuration from the list.  
If you did not enter your log-on information on the **New Server Connection** page, the **Log On** dialog box opens.
2. Enter a **User name** and **Password** in the appropriate text boxes.  
Passwords are case sensitive and may have length restrictions.
3. *Optional:* Check **Save as default credentials for this server**.
4. Click **OK**.

### Designating a Project View and Folder to Share


1. Expand the nodes of your server configuration to select a project view.
2. Click **Next**.
3. Do one of the following:
  - Select a folder from the resulting list.
  - Click **New Folder** to create one.
4. Click **Next**.  
By default, this setting maps to the default setting for StarTeam server-side (remote) folders. Most users will accept this as their default setting. This means that you are using the default folder name provided by StarTeam Server to name your local folders. However, while sharing the Eclipse project with StarTeam, you can override this setting at the project-level by checking **Enable project-specific settings**.
5. Choose desired settings in the Mapping Preferences page, or accept the default settings, and click **Finish** . A dialog box opens asking you if you would like to synchronize the newly-shared project.

### Synchronizing Newly-Shared Projects

1. Click **Synchronize**. The **Team Synchronizing** perspective opens. The **Incoming/Outgoing** mode is automatically selected showing the files to add to the repository.
2. Select the appropriate files or folders that you wish to check in, right-click, and choose **Check In**. The **Check In** dialog box opens.

### Opening an Existing Project View

Before you can access an existing StarTeam project view, you must log onto the appropriate server configuration to display the projects and views associated with it.

 **Important:** The term **view** should not be confused with the Eclipse UI usage of **view**. In StarTeam, a view consists of an application folder tree and the items associated with each folder in that hierarchy.

1. Open Server Explorer.
2. Right-click the name of the StarTeam server configuration on which the project is located, and choose **Log On**. The **Log On** dialog box opens.
3. Type **User name** and **Password**.
4. Click **OK**.
5. Expand the server configuration name to display the list of projects for the selected server configuration.
6. Expand the project you wish to open. Displays the name of the project root view.
7. Expand to the folder level.


## Create Projects and Synchronize Local Files

At a high-level, creating a new StarTeam project and associating an existing workspace project with it involves the following steps:



These steps assume that:


- You have not yet created a connection to a StarTeam server configuration.
- You have already created an Eclipse project in the Workspace to share.

### Creating Projects

1. Open a StarTeam perspective or view.
2. Choose **StarTeam > Project > New**. The **New Project Wizard** opens.
3. Select a server configuration for the project from the **Server** list.  
 **Note:** If the server you want is not in the **Server/Project Tree**, click **Add Server** and add it using the **Add Server** dialog box.
4. Click **Next** to continue.
5. Type the name and description for the new project in the **Project Name** and **Project Description** fields, and click **Next**.
6. Browse to the folder on your computer that will be the default working folder for the project root folder.

If the working folder does not exist, type the folder path and name and the **New Project Wizard** will create it.

-  **Note:** The default working folder must point to a location that is physically discrete for each user, such as a drive on your local computer or a personal directory on a shared file server.
7. Click **Next**.
8. Optionally, if the working folder has child folders, select any child folders you do not want added to the project and click **Exclude**.  
 **Tip:** Click **Reset** to include the previously excluded folders.
9. Click **Finish** to complete and open the project.

 **Note:** After you create a project, the Server Explorer window displays a hierarchical **Folder Tree** of folders in the current view of the project. You can add other folders, if desired.

### Sharing Workspace Projects with New Project

1. Select your workspace project in the Package or Navigator Explorers, and choose **Team > Share Project**.
2. Select **StarTeam**.
3. Click **Next**.
4. Select your server configuration containing the newly-created StarTeam project and view.

5. Click **Next**.
6. Select a folder from the list, or click **New Folder** to create one.
7. Click **Next**. The Mapping preferences page opens. By default, this setting maps to the default setting for StarTeam server-side (remote) folders. Most users will accept this as their default setting. This means that you are using the default folder name provided by StarTeam Server to name your local folders. However, while sharing the Eclipse project with StarTeam, you can override this setting at the project-level by checking **Enable project-specific** settings.
8. Choose desired settings in the Mapping Preferences page, or accept the default settings, and click **Finish**. A dialog box opens asking you if you would like to synchronize the newly-shared project.

## Deleting Projects

To delete a project, you must have the delete privilege or access right. Be absolutely certain that you want to delete a project and its folders because you and other members of your team will no longer be able to access any item in the project.

If other users are connected to the project at the time it is deleted, they will receive a message the next time they initiate a project or view command.

Deleting a project does not remove any data from the StarTeam Server database. However, items that are not shared will no longer be accessible.

1. Open the Server Explorer.
2. Right-click on the project name and choose **Delete**.
3. A message displays asking you to confirm the deletion. Click **OK**.
4. A dialog box opens requesting you to enter the exact name of the project that you wish to delete. Type the name in the field provided.
5. Click **OK**.

## Changing Project Names or Descriptions

If you have the appropriate access rights, you can use the **Properties** dialog box to review or change the project name and description.

1. Choose **StarTeam > Project > Properties**. In the Server, Navigator, or Package Explorers, right click on the project, and choose **Properties** from the context menu. The **Properties** dialog box opens.
2. Expand **StarTeam Provider** and select **Project**.
3. Click the **Name** tab.
4. Type a new project name in the **Name** field.
5. Type a new project description in the **Description** field.
6. Click **OK**.

## Refactoring

"Team refactoring" involves renaming or moving files and folders that have been placed under version control in a shared environment.



**Note:** When conflicts occur, the client provides immediate feedback, and a dialog box opens asking you to confirm the change.

1. In the Navigator or Package Explorer, select an item.
2. Right-click and choose either:
  - **Move**
  - **Rename**
3. Do one of the following:

- For a move, select the destination on the **Folder Selection** dialog box, and click **OK**.
- For a rename, type the new name, and press **Enter**.



**Note:** Before renaming a folder, click **F5** to perform a refresh in Eclipse.

If you use the **Move** command on your project items and you try to check in your changes, Eclipse reports this and suggests that you use the **Move** action in the **Synchronize** view to preserve the resource history.

4. Open the **Team Synchronizing** perspective's **Synchronize** view to review any items reflecting a *[moved]* status.
5. Multi-select any *[moved]* items in the **Synchronize** view and choose **Move**.

## Viewing or Modifying Project Properties

Each project can be configured with properties that allow you to specify the project name, enable keyword expansion, require revision comments on check-in, specify file locking behavior on check-in, define process rules and process items, and specify alternate property editors.

1. Choose **StarTeam > Project > Properties** . In the Server, Navigator, or Package Explorers, right click on the project, and choose **Properties** from the context menu. The **Properties** dialog box opens.
2. Expand **StarTeam Provider** and select **Project**.
3. Select a tab containing properties you want to view or change and make the changes.
4. Click **OK**.

## Enabling Keyword Expansion

By enabling keyword expansion for a project, you can embed keywords within text files. These keywords are automatically expanded during file check-outs, to provide file and revision information within the file. You should use only one keyword per line.

Keyword expansion and EOL conversion work correctly for UTF-8 unicode files, which previously could be corrupted on check-in. However, EOL conversion is not supported for UTF-16 or UTF-32 unicode files. .

StarTeam supports Log and History Keywords from the StarTeam Server and from the MPX Cache Agent.



**Caution:** Never use a keyword in a revision comment, as it will be expanded during the keyword expansion process.

1. Choose **StarTeam > Project > Properties** . In the Server, Navigator, or Package Explorers, right click on the project, and choose **Properties** from the context menu. The **Properties** dialog box opens.
2. Expand **StarTeam Provider** and select **Project**.
3. Click the **Options** tab.
4. Check the **Keyword Expansion** check box to enable keyword expansion and use keywords in your text files.

This check box applies to files added or checked in from StarTeam or from StarTeam integrations with third-party applications.

5. In **Expand Keywords for These File Extensions** field, type the file extensions (for example, `.bat`, `.cpp`) for which you want to use keywords.

You can use a space, comma, or semicolon as keyword delimiters. The file extensions list can contain a maximum of 254 characters. If you leave this text box blank, no keywords will be expanded.



**Note:** When Keyword Expansion is enabled, the StarTeam Eclipse Plugin uses the encoding specified in the StarTeam **Properties** dialog box to expand the keywords in files as the files are checked-out. After check-out, any encoding changes are managed by Eclipse, not the StarTeam Eclipse Plugin.

## StarTeam Keywords

By enabling keyword expansion for a project, you can embed keywords within text files. These keywords are automatically expanded during file check-outs, to provide file and revision information within the file. You should use only one keyword per line.

Item	Description
<code>\$Author\$</code>	User who checked in the revision.
<code>\$Date\$</code> and <code>\$DateUTC\$</code>	Date and time stamps for the revision. <code>\$DateUTC\$</code> is the same as <code>\$Date\$</code> except that a UTC time replaces the local time. UTC times end in a "Z," which makes them readily identifiable.
<code>\$Header\$</code> and <code>\$HeaderUTC\$</code>	<p>Combinations of Workfile, Revision, Date, and Author. <code>\$HeaderUTC\$</code> is the same as <code>\$Header\$</code> except that a UTC time replaces the local time. UTC times end in a "Z" which makes them readily identifiable.</p> <p>For Java users, <code>\$Header\$</code> can cause problems if <code>\u</code> (for unicode) appears in the expanded header. For example, suppose that a file named <code>foo.java</code> is stored in <code>D:\util</code>. The first time you compile it with <code>\$Header\$</code>, the header is expanded to:</p> <pre>\$Header: D:\util\foo.java, 1, 7/27/11 11:05:48 AM, StarTeam Server Administrator\$</pre> <p>Even though this header is contained in a Java comment, the Java compiler always looks for <code>\u</code>. The second time you compile <code>foo.java</code>, a compiler error occurs.</p>

### `$History$`

Added to the source file, typically within a comment field. The `$History$` keyword creates a history record for the latest (tip) revision and places the information after the keyword. For example, after the file is checked-out for the first time, the file would contain the following:

```
// ...
// $History
// 1 YourProject 1.0 2011-11-19 00:06:57Z Joe Smith
// This is a revision comment.
// $
```

StarTeam adds the history information to the file and places it outside of the "\$" signs so that it becomes a versioned part of the file. Since the history records become part of the versioned file content, you can delete extra or excessive history records at any time. If this file is modified and checked-in, any subsequent check-out adds one additional record immediately after the `$History$` keyword:

```
// $History
// 2 YourProject 1.0 2012-06-01 00:06:57Z Joe Smith
// This is a another revision comment.
// $
// 1 YourProject 1.0 2011-11-19 00:06:57Z Joe Smith
// This is a revision comment.
```

StarTeam uses the MD5 and local file timestamp of the post-expanded file for sync records. Accordingly, when you check-out a tip revision, StarTeam reports a *current* status for the file even though a compare tool would show an additional history record in the locally-stored file as compared to the version of the file saved on the StarTeam Server.

**Important:** Do not delete the standalone `// $` line. StarTeam places the most recent historical revision information within the `// $` delimiter. The rest of the revision entries are outside of this delimiter. StarTeam does not recognize the `$History$` keyword if you remove this line.

Item	Description
<code>\$Id\$</code>	Similar to <code>\$HeaderUTC\$</code> except that it is a combination of Workfile, the branch revision number (preceded by "v" for version; for example, v 1.2.1.0), UTC time, and Author. The branch revision number is in dot notation.
<code>\$Locker\$</code>	User who has the file exclusively locked (if any).
<code>\$Log\$, \$Log[x]\$, \$LogUTC\$, and \$LogUTC[x]\$</code>	<p>File change history. <code>\$Log\$</code> is a special keyword because it expands to a multiline entry. The <code>\$Log\$</code> keyword expands to include information for each revision of the file. Revision history includes Revision Number, Date, Author, and Reason for Check In.</p> <p>Use <code>\$Log\$</code> to retain entries for each revision within the file.</p> <p>Use <code>\$Log[x]\$</code> to limit the number of revisions for which entries are retained. Replace <code>x</code> with the number of entries to be retained. For example, <code>\$Log[8]</code> saves the entries for the most recent 8 revisions. If you replace <code>x</code> with a number less than 1 or with a non-numeric character, StarTeam ignores <code>x</code> and retains all entries (as with <code>\$Log\$</code>).</p> <p><code>\$LogUTC\$</code> and <code>\$LogUTC[x]\$</code> are the same as <code>\$Log\$</code> and <code>\$Log[x]\$</code> except that a UTC time replaces the local time. UTC times end in a "Z".</p>
<code>\$NoKeywords\$</code>	Turn off keyword expansion for the rest of the file.
<code>\$Project\$</code>	Name of the project.
<code>\$Revision\$</code>	Revision number (an integer).
<code>\$Folder\$</code>	Name of the folder.
<code>\$Workfile\$</code>	Unqualified name of the working file (for example, <code>foo.cpp</code> ).



**Caution:** Never use a keyword in a revision comment, as it will be expanded during the keyword expansion process.

## Establishing Process Rules for Projects

Establishing a system of process rules allows you to:

- Require that process items are used every time files are added or checked into the project.
- Stipulate that only certain types of items with specific statuses can be used as process items in the project.
- Enable the use of enhanced process links for the project.




**Note:** To set process rules, you must have the access rights required to change project properties. Usually, only team leaders and administrators have these rights. You must also verify that project users have the rights to see and modify items in the project view, to create and modify links on files and process items, and to create tasks and link to tasks if using the enhanced model.

1. Choose **StarTeam > Project > Properties**. In the Server, Navigator, or Package Explorers, right click on the project, and choose **Properties** from the context menu. The **Properties** dialog box opens.
2. Expand **StarTeam Provider**, and select **Project**.
3. Select the **Require Selection Of Process Items When Files Are Added Or Checked In** check box.
4. Select the type you want to allow for use as process items.
5. You can define the use of the type as a process item in the **Process Item Details for <Type>** section.

To permit the use of any type as a valid process item:




1. Select the desired **Type**.
2. Specify the **Active States** that are permitted to be used as a process item during commit.
3. Specify the **Closed State** that will be used to mark the process item as completed upon successful check-in.
4. Add the <Type> as a valid process item type.

 **Note:** Some StarTeam integrations do not recognize process rules and will ignore them.


## Setting Active Process Items

Setting an active process item is a convenient way of saving time when you know that you will be adding files or checking them in later.


You can make a selected change request, task, or requirement the active process item for the current view, an open view on the same server, or a different view on the same server.

 **Note:** You can only specify one active process item for each view. Setting a second active process item for the same view at one time clears the first one.

1. Open your project in a StarTeam perspective.
2. Click the **Change Request**, **Requirement**, or **Task** tab and select the item you want to use as the active process item.
3. Do one of the following:
  - Right-click and choose **Set Active Process Item > Current View** to choose the current view.
  - Right-click and choose **Set Active Process Item > [view name]** to choose from the listed opened views on the server.
  - Right-click and choose **Set Active Process Item > Select View** to open a dialog box and choose any other view on the server.

 **Note:** You can also set the currently-selected item as a process item by using the **Use As Active Process Item** button on the toolbar.

The active process item you selected is used by default when you add files or check them in. However, you can override this default and select another appropriate item when adding or checking in files.


 **Tip:** After you finish with a process item, you should right-click it and choose **Clear Active Process Item** so that it cannot be accidentally reused. That removes the information from the status bar and keeps the process item from reappearing in the **File Add** or **File Checkin** dialog boxes.

## Excluding Files from a Project

Some types of files will never be added to a project, although they may reside in a working folder. For example, suppose you are creating files with an application that makes an automatic backup (.bak) copy of each file every time you save the file. Although your working folder might contain several .bak files, you would have no reason to check them into (or out of ) the application. Therefore, you should exclude them from the project view.

Exclude lists can also be inherited from parent folders.

1. With a StarTeam perspective or view open, choose **StarTeam > Folder > Properties**.
2. Select **Folder**.
3. Select the **Exclude** tab.

 **Note:** The **Exclude** tab does not affect files that are already part of the project.

4. Do one of the following:

<b>Inherit and Use Local Exclude List</b>	Excludes files that match the exclude list specifications set for this folder as well as those of its parent folder. If the <b>Local Exclude List</b> text box does not yet include any file specifications, add them.
<b>Use Local Exclude List</b>	Excludes files that match the exclude list specifications set for this folder. If the <b>Local Exclude List</b> text box does not yet include any file specifications, add them.
<b>No Exclude List</b>	Includes all files.

5. Type one or more file specifications to use for matching files.

Use standard expressions (with \* and ? wild cards) separated by commas, spaces, or semicolons. To include a comma, space or semicolon as part of the specification, enclose the specification in double quotes.

A trailing / character means that **Not-in-View** folders will be excluded. For example, bin/ would cause all **Not-in-View** folders named bin to be excluded from the folder tree.



**Note:** The \ character does not work. It is treated as an escape character.

## Requiring Exclusive Locks for Check-ins

With this application, you cannot force users to lock files before they make changes. However, if you have the required privileges, you can require all users to conform with the policies and processes of your company by exclusively locking files before they check them back into the application.

Although this requirement helps users to avoid merge situations, they still must:

- Notice whether a file is already exclusively locked by another user before they check it out to work on it.
- Lock each file before making changes, to alert other users to their intentions.
- Be sure that the status for each working file is `Current` to avoid changing older revisions of the files. If a file status is not `Current`, the file must be checked out before any changes are made.

To force users to lock files before checking them in

1. Choose **StarTeam > Project > Properties** . In the Server, Navigator, or Package Explorers, right click on the project, and choose **Properties** from the context menu.

The **Properties** dialog box opens.

2. Select **Project**.
3. Click the **Options** tab.
4. Check **Require Exclusive Lock When Files are Checked In**.

When this option is selected, only a person who has exclusively locked a file can check it in.



**Note:** If developers are using an application integration for a development environment, such as StarTeam Visual Studio Plugin, they cannot check in files from that environment if both the **Require Exclusive Lock When Files are Checked In** check box in the **Project Properties** dialog box, and the **Use Non-exclusive Locks in Integrations** check box on the **Personal Options** dialog box (**File** tab) are selected. In this situation, uncheck **Use Non-exclusive Locks in Integrations** to check files in.

## Requiring Revision Comments

When users check in files by using **File > Check In** or selecting **Check In** from the context menu, the **Check In** dialog displays. By default, this dialog allows them, but does not require them, to type a comment about the operation.

If users check in files by clicking either of the **Check In** buttons on the toolbar, this dialog box does not open, so they cannot type a revision comment.

Administrators can force users to supply a check-in reason, however, by adjusting the properties for the project. This requirement will apply no matter which way users perform the check-in.

1. Choose **StarTeam > Project > Properties** . In the Server, Navigator, or Package Explorers, right click on the project, and choose **Properties** from the context menu.

The **Properties** dialog box opens.

2. Select **Project**.
3. Click the **Options** tab.
4. Check **Require Revision Comment When Files are Checked In**.



**Note:** This check box applies only to the application, not to integrations.

5. Click **OK**.

From this time on, the **Check-in** dialog box will always open when users check in files, and they will need to type text in the **Reason for Check-in** field before completing the operation.

## Marking Unlocked Files Read-only

In many cases, users make edits before realizing that their files must be exclusively or non-exclusively locked to check them in. If the files are read-only, users are less likely to make this mistake.

1. Choose **StarTeam > Project > Properties** . In the Server, Navigator, or Package Explorers, right click on the project, and choose **Properties** from the context menu..

The **Properties** dialog box opens.

2. Select **Project**.
3. Select the **Options** tab.
4. Check **Mark Unlocked Working Files Read-only**, which applies to files that are unlocked in the application or in application integrations with third-party applications.

If this check box is cleared, you must use the operating system to change the read-only attribute to read/write.

Working copies of unlocked files will now become read-only when the following file operations are performed:

- File check-ins.
- File check-outs (from the **File** or **History** pane).
- File unlocks.



**Note:** This project property overrides the identical **Mark Unlocked Working Files Read-only** personal option. If you change your mind after selecting the property (or the equivalent personal option), verify that no files are writable before clearing the check box. Next, force a check out and lock all the files (or just the read-only files). Finally, unlock them.

## Searching Projects

1. Open a StarTeam perspective or view.
2. Click **Search for StarTeam Items**.  
Alternative: you can choose **Search > Search** from the menu.  
The **Search** dialog box opens.
3. Select the **StarTeam Search** tab.
4. Type the text for which you want to search into the **Regular expression** text box.
5. In the **Search in** area of the dialog box, select any desired server, project, view, or folder that you wish to search in.

6. Use the **Type** list to narrow your search to specific StarTeam item types and properties. Select **Files** in from the list and then choose the **Comments** property from the **Look in** options to search Comments included with Files only.

7. From the **Look in** options, choose one of the following:

**All properties** Returns all item properties. An item is accepted if the string representation of the value of any of the item properties matches the given regular expression.

**All properties by current filters** Processes only properties in the current filter that are applicable to the chosen type.



**Note:** you can see the list of these properties by opening the **This property** list box when **Choose from all properties by current filters** is selected.

**This property** Select to specify a single property to be processed.

8. From **Options**, you can select:

**Match case** Returns only items that match the case as entered into the **Regular expression** field.

**Recursive** Seeks matches in all folders beneath the selected items from **Look in**.

9. Click **Search**.

The results will be displayed in the **Search** view and on the **StarTeam Search Results** tab.

## Synchronization

Projects on the StarTeam server and projects in the Eclipse workspace can be shared. The client sits on top of and supports the standard Team Synchronization perspective in Eclipse. All incoming and outgoing changes within shared projects are synchronized on a regular basis with the StarTeam repository. In Eclipse terminology, synchronizing outgoing changes sends all changes and new files created in Eclipse to StarTeam Server, while synchronizing incoming changes brings all changes and new files from StarTeam Server into Eclipse.

The client provides StarTeam perspectives and views and menu commands added to the standard **Team** context menu. The **Team** context menu is surfaced throughout the Eclipse UI and provides the default **Team** menu commands and several StarTeam-specific commands enabling you to work in a team environment with StarTeam as your source control provider. Using the StarTeam perspectives and views along with the **Team** context menu commands provides the quickest route to working with your StarTeam source control items.

Additionally, the Team Synchronizing perspective provided in Eclipse allows you to review the current status of each conflict and resolve the majority of problems.

### Synchronizing with the Repository

1. Do one of the following:

- Select an item from the Navigator or Package Explorers, and choose **Team > Synchronize with Repository**.
- Open the Team Synchronizing perspective and choose **Window > Open Perspective > Team Synchronizing**.

The **Team Synchronizing** perspective opens.

2. In the **Synchronize** view, select the **Incoming/Outgoing** mode.


3. Select the appropriate files or folders that you wish to perform operations on.

4. Use the context menu commands in the **Synchronize** view to perform check in or check out operations.

## Synchronizing Newly-Shared Projects

1. Click **Synchronize**. The **Team Synchronizing** perspective opens. The **Incoming/Outgoing** mode is automatically selected showing the files to add to the repository.
2. Select the appropriate files or folders that you wish to check in, right-click, and choose **Check In**. The **Check In** dialog box opens.

## Copying URLs

 **Note:** StarTeam shortcuts use the file extension `.stx`.

1. Select one or more items in an item view or the Server Explorer.
2. Right-click the selected items and choose **Copy URL to clipboard**. This action places in the clipboard a plain text version of the URL to the selected items and an HTML representation of the links to the selected items. From the clipboard, you can paste the URL to a selected application.
3. Paste the URL to the application of choice.  
Copying a URL to the clipboard is equivalent to dragging an item or items from the list pane or folder tree onto an application. Not all applications support pasting the HTML representation, although Word, Excel, and Outlook do support HTML data.

## Project Properties

This section presents the StarTeam project properties and their descriptions as displayed in the **Properties** dialog box.

### Name (Project Properties Dialog Box)

**StarTeam > Project > Properties > Name**

Describes the options on the **Name** page of the **Properties** dialog box.

<b>Name</b>	Specifies the project name.
<b>Description</b>	Specifies the project description.
<b>Created By</b>	Displays the name of the person who created the project.
<b>(Created) On</b>	Displays the date on which the project was created.
<b>Type</b>	Displays the project type.

### Options (Project Properties Dialog Box)

**StarTeam > Project > Properties > Options**

Describes the options on the **Options** page of the **Properties** dialog box.

<b>Keyword Expansion</b>	Enables keyword expansion.
<b>Expand Keywords For These File Extensions</b>	Restricts keyword expansion to files with the specified extensions.
<b>Require Revision Comment When Files Are Checked In</b>	Specifies that revision comments are required when checking in files to the project.
<b>Require Exclusive Lock When Files Are Checked In</b>	Specifies that all files checked in the must be exclusively locked to enable check-in.
<b>Mark Unlocked Working Files Read-Only</b>	Specifies that all unlocked working files are marked as read-only if they are checked in, checked out, or unlocked when file locking is required.

This option applies to files that are unlocked in the application or in application integrations with third-party applications.

The project property overrides the identical **Mark Unlocked Working Files Read-only** personal setting.

## Process Rules (Project Properties Dialog Box)

### StarTeam > Project > Properties > Process Rules

Describes the options on the **Process Rules** page of the **Properties** dialog box.

#### Require Selection Of Process Items When Files Are Added Or Checked In

Enforces selection of process items when files are added or checked in.

#### Permit Selection Of Change Requests As Process Items

Permits the selection of change requests as process items, and restricts them based on which status fields are checked in the **Restrict Status To** check boxes:

- If **Open** is checked, only change requests with a status of **Open** can be used as process items.
- If **In Progress** is checked, only change requests with a status of **In Progress** can be used as process items.
- If both **Open** and **In Progress** are checked, only change requests with a status of **Open** or **In Progress** can be used as process items.
- If neither **Open** and **In Progress** are checked, all change requests can be used as process items, regardless of their status.

#### Permit Selection Of Requirements As Process Items

Permits the selection of requirements as process items, and restricts them based on which status fields are checked in the **Restrict Status To** check boxes:

- If **Approved** is checked, only requirements with a status of **Open** can be used as process items.
- If **Approved** is unchecked, all requirements can be used as process items, regardless of their status.

#### Permit Selection Of Tasks As Process Items

Permits the selection of tasks as process items, and restricts them based on which status fields are checked in the **Restrict Status To** check boxes:

- If **Ready To Start** is checked, only tasks with a status of **Ready To Start** can be used as process items.
- If **In Progress** is checked, only tasks with a status of **In Progress** can be used as process items.
- If both **Ready To Start** and **In Progress** are checked, only tasks with a status of **Ready To Start** or **In Progress** can be used as process items.
- If neither **Ready To Start** and **In Progress** are checked, all tasks can be used as process items, regardless of their status.

#### Enable Enhanced Process Links

Enables the client to use enhanced process links.

If checked, the process item (that is, the item specified as the reason for making a given set of changes) is distinguished from the task that represents the act of making the associated changes in a particular view. Changes are linked to the process item indirectly, through a workspace (check-in) change package which is automatically created by the client.

If unchecked, standard links are used where the source of a process link is itself a process item. That is, if a given item is specified as the reason for a change, then process links are created directly from that process item to each changed file or folder.

## Editors (Project Properties Dialog Box)

StarTeam > Project > Properties > Editors

The check box options on the **Editors** page of the **Properties** dialog box.

<b>Use Alternate Property Editor For Files</b>	Specifies that for files, StarTeam should use the specified alternate property editor in the corresponding field.
<b>Use Alternate Property Editor For Change Requests</b>	Specifies that for change requests, StarTeam should use the specified alternate property editor in the corresponding field.
<b>Use Alternate Property Editor For Requirements</b>	Specifies that for requirements, StarTeam should use the specified alternate property editor in the corresponding field.
<b>Use Alternate Property Editor For Tasks</b>	Specifies that for tasks, StarTeam should use the specified alternate property editor in the corresponding field.
<b>Use Alternate Property Editor For Topics</b>	Specifies that for topics, StarTeam should use the specified alternate property editor in the corresponding field.



**Note:** Your company must use StarTeam Enterprise Advantage to be able to use APEs. For more information creating APEs, refer to the *StarTeam Extensions User's Guide*.

## Provider Properties

This topic presents the StarTeam **Provider** properties and their descriptions as displayed in the **Provider Properties** dialog box when opened from the context menu. The information in the **Provider Properties** dialog box is different for each type of selected, shared resource: project, folder, or file.

### Shared Project

The following properties are displayed for the shared project.

<b>Server Description</b>	Displays the name of the StarTeam Server on which the shared project is located.
<b>Project Name</b>	Displays the name of the selected shared project.
<b>View Name</b>	Displays the name of the view which contains the shared project.
<b>Folder</b>	Displays the folder on the StarTeam Server that is used for sharing.
<b>Change Sharing</b>	Opens the <b>Share a Project</b> dialog box where you can change the sharing options.
<b>Change View</b>	Opens the <b>Select a StarTeam View</b> dialog box where you can change which view you are sharing.
<b>Change Kind</b>	Reveals the types and directions of changes that can be found in any resource within the project. If it reads <b>[in sync]</b> , there are no files and folders that are not "current".
<b>Base Bytes</b>	Shows both the dot-notation and the ID for the <b>StarTeam</b> folder used for sharing.
<b>Remote Bytes</b>	Shows the same as Base Bytes, but reflects the current situation on the StarTeam Server. It is possible for the dot-notation to be higher, indicating a newer version is available in the repository.

## Shared Folder

The following properties are displayed for the shared folder.

<b>Server Description</b>	Displays the name of the StarTeam Server on which the shared project is located.
<b>Project Name</b>	Displays the name of the selected shared project.
<b>View Name</b>	Displays the name of the view which contains the shared project.
<b>Folder</b>	Displays the folder on the StarTeam Server that is used for sharing.
<b>Change Kind</b>	Reveals the types and directions of changes that can be found in any resource within the project. If it reads <b>[in sync]</b> , there are no files and folders that are not "current".
<b>Base Bytes</b>	Shows both the dot-notation and the ID for the StarTeam folder used for sharing.
<b>Remote Bytes</b>	Shows the same as <b>Base Bytes</b> , but reflects the current situation on the StarTeam Server. It is possible for the dot-notation to be higher, indicating a newer version is available in the repository.
<b>Convert Change Kind To</b>	Allows you to invert the direction and type of a deletion or an addition. This is sometimes necessary when a project was re-shared or hooked up with another view. Changes on the StarTeam Server or in the workspace may be interpreted assuming the wrong intention. A folder listed as <b>[outgoing deletion]</b> may simply have been added to the view by another developer at some time, and thus be considered an <b>[incoming addition]</b> . The two buttons, <b>Incoming Deletion</b> and <b>Outgoing Addition</b> , allow you to change

## Shared File

The following properties are displayed for the shared file.

<b>Server Description</b>	Displays the name of the StarTeam Server on which the shared project is located.
<b>Project Name</b>	Displays the name of the selected shared project.
<b>View Name</b>	Displays the name of the view which contains the shared project.
<b>Folder</b>	Displays the folder on the StarTeam Server that is used for sharing.
<b>MD5:</b>	Displays the current MD5 hash value for the local revision.  <b>Stored Remote MD5:</b> Displays what the provider remembers the MD5 hash value to have been at the last check in or check out.  <b>Current Remote MD5:</b> Displays the MD5 hash value for the tip revision. By comparing the stored MD5 hash value with the current MD5 hash value, the provider can detect incoming changes in spite of differing remote and local MD5 values due to EOL differences.
<b>Base Modification Timestamp:</b>	Shows the last date and time the local and remote files were current.  <b>Local Modification Timestamp:</b> Shows the date and time of the last modification to the local file.  <b>Remote Modification Timestamp:</b> Shows the date and time of the last modification to the remote file.
<b>Change Kind</b>	Reveals the types and directions of changes that can be found for any file resource within the project. If it reads <b>[in sync]</b> , there are no files and folders that are not "current".



<b>Base Bytes</b>	Shows both the dot-notation and the ID for the StarTeam file used for sharing.
<b>Remote Bytes</b>	Shows the same as <b>Base Bytes</b> , but reflects the current situation on the StarTeam Server (tip revision). It is possible for the dot-notation to be higher, indicating a newer version is available in the repository.
<b>Convert Change Kind To</b>	Allows you to invert the direction and type of a deletion or an addition. This is sometimes necessary when a project was re-shared or hooked up with another view. Changes on the StarTeam Server or in the workspace may be interpreted assuming the wrong intention. A folder listed as <b>[outgoing deletion]</b> may simply have been added to the view by another developer at some time, and thus be considered an <b>[incoming addition]</b> . The two buttons, <b>Incoming Deletion</b> and <b>Outgoing Addition</b> , allow you to change.
<b>Stored EOL Conversion</b>	Specifies which EOL (End of Line) conversion settings to store. When you click <b>OK</b> or <b>Apply</b> after changing this setting, the MD5 values will be recomputed and the status refreshed in an attempt to identify the type of change.



**Note:** StarTeam uses MD5 hash values exclusively for status computation, therefore it needs to remember which EOL conversion was in effect at the last check in or check out so it can adjust the computed MD5.

## View Configuration and Management

A view is a window through which you can access a subset of the artifacts in a project. Furthermore, a view uses items to reference artifacts and manage access to them. Depending on how it is configured, a view can:

- Serve as a sub-project for a specific development or maintenance activity.
- Be a read-only or updateable subset of another view.
- Be used for other purposes.

The good news is that views have tremendous flexibility to serve many needs. The bad news is that you can get unexpected or undesirable results if you don't understand how the different view types work.

This section describes each view type, how it behaves, and when you might want to use it. It also discusses the roles views can fulfill for specific development activities and suggests practices for managing changes within a view and for propagating changes from one view to another.

## Overview of Views

When you create a new project, the server creates an initial or root view of that project with only the "File" type pre-selected as a default for new views. Users can still change the project properties after the project is created, and they can change the item types included for any given new view. However, if the user changes nothing, by default new views will only include files when they are created.

This initial view, which has the same name as the project, consists of the root folder, to which you will add child folders, files, and eventually, more. It is always read/write. The root view is ideal for collaborative development, because it is dynamic, showing all items in the project as they change.

To accommodate both user and project needs, however, the application enables you to create additional views of a project based on this view. These additional views, called child views, may contain some or all of the contents of the original view and may behave differently. For example, you might use child views to:

- Implement the same folder hierarchy for multiple releases of a product. If the root view is for current development, you maintain the folder hierarchy there and create new child views for each release at about the time that it ships. Maintenance would be done in the child view.
- Limit the portion of the project that certain team members see. Developers might need to see only the project's source code folder and its child folders; marketing personnel might need to see only the

marketing folder and its child folders; and so on. Each of these views can have a different folder as its root.

- Support branching and parallel development. By branching files and other data in a new view, your organization can start to work on the 2.0 version of a product without hampering the creation of service packs for the 1.0 version.

Views represent configurations of items and support different development baselines of the same code base. It is common practice to promote changes from one release's maintenance view to the root view where the next release is being created. Views can be reconfigured to show items as they existed at an earlier point in time, or based on a view label or associated promotion state. Rollback views are read-only, as they show a precise state of the items and no longer permit changes.

## Understanding Branching Views

A branching view is a view that permits branching—that is, the folders and other items in the view can branch or separate from the corresponding items in the parent.

Branching views serve many purposes; you can create a branching view to meet different needs from those of your main line of development. For example, you might create a branching view for a maintenance release or a custom version of your product. A branching view can also be used to keep an area of your project private until it is completed and tested. Then you can merge your changes into the main line of development when and where necessary.

A branching view should use a different working folder than that of its parent view. Using the same working folder for both views is not only confusing but can create status problems.

## Item Branching Behavior

Given the appropriate settings, folders, files, and change requests in a child view can be branched, that is, can be separated from the corresponding item in the parent view. Folders and change requests branch when their properties change, while files branch when either their contents or their properties change (Requirements, tasks, and topics can never branch).


For each item, branching occurs a maximum of one time per view. For example, if a new item is added to the root view, its first revision has the dot notation 1.0. Subsequent revisions become 1.1, 1.2, and 1.3. Suppose this item is included in two child views created from the root view and that both of the child views are branching views. In one child view, if a new revision is made to the item, the item branches. This separation from its "parent" item in the root view is indicated by an addition of two numbers to the dot notation. If the parent item was 1.3, the child item becomes 1.3.1.0. The child item's next revision becomes 1.3.1.1.

Now, suppose the corresponding item in the second child view is changed. Its dot notation must also change. Because 1.3.1.0 already exists, the separation of this item from its parent item gets the dot notation 1.3.2.0. This child item's next revision becomes 1.3.2.1.

The original item in the root view has the history: 1.0>1.1>1.2>1.3. The next revision will be 1.4. The item in one child view has the history: 1.0>1.1>1.2>1.3>1.3.1.0>1.3.1.1. The next revision will be 1.3.1.2. The item in the other child view has the history: 1.0>1.1>1.2>1.3>1.3.2.0>1.3.2.1. The next revision will be 1.3.2.2.

Whether or not a folder, file, or change request has the ability to branch depends upon its behavior settings.

- If the **Branch on Change** check box for an item is enabled and selected, the item can branch.
- If the **Branch on Change** check box for an item is enabled but not selected, the item cannot branch, but its behavior can be changed.
- If the **Branch on Change** check box for an item is disabled, the item has either already branched or its location is where it was added to StarTeam.
- If the **Branch on Change** check box for a folder is enabled but not selected, new items cannot be added. If you attempt to do so, an error message will appear, stating that the folder is read-only.

 **Note:** Changing a folder's branching behavior does not affect the behavior of items in the folder. Items in a folder have their own branching behavior.

### Items in Both the Parent and the Branching View

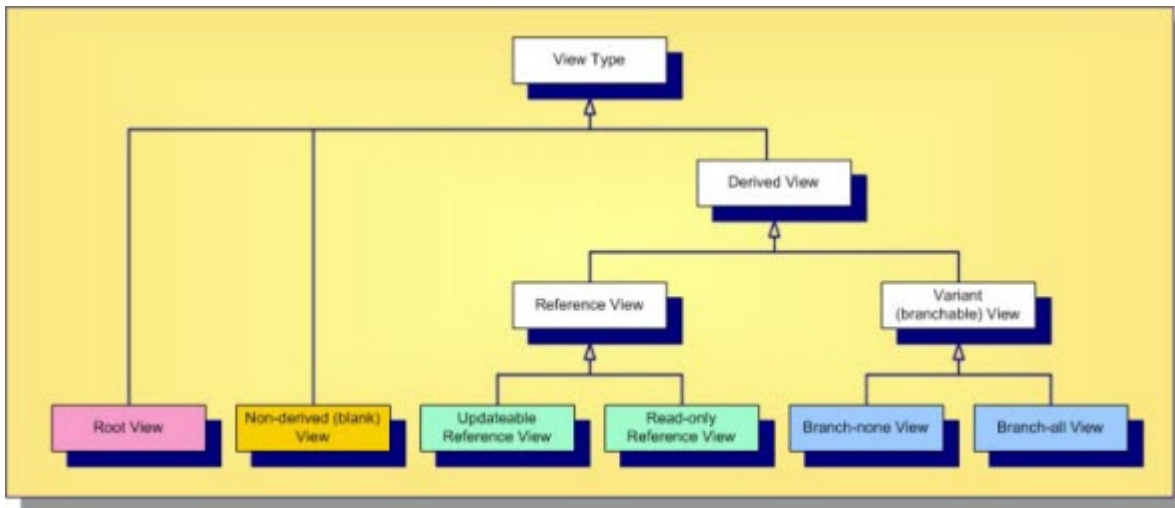
Depending on the folders included in the new branching view, certain items from the parent view appear in the new branching view. However, these “inherited” items will have no labels, as neither view labels nor revision labels move from view to view.

As a result, the workflow for change requests is affected in the following ways:

- If the **Last Build Tested** and the **Addressed In Build** fields in a change request have no values when the change request branches, their workflow is specific to the new view only.
- If the **Last Build Tested** and the **Addressed In Build** fields in a change request have build labels as their values (i.e., these fields are not empty or do not contain the value `Next Build`) when the change request branches, the branched change request retains those values. In the new view, these values can be changed, but only to the names of build labels that exist in the new view.
- If the **Addressed In Build** field contains the value `Next Build` when the change request branches, the `Next Build` value is replaced by the name of the next build label created in the parent view, not the next build label created in the new view.

## View Types

As shown, there are six concrete view types. Aside from the special *root* view type and one *non-derived* view type, the rest are *derived* view types, which subdivide into *reference* and *variant* types, each of which have two concrete types.



The abstract and concrete view types are described below.

**Root View** A root view is the “main” view automatically created for each project. Initially, it receives the same name as the project, but you can change it. (It is often renamed “Main” to emphasize its role.) When first created, the root view has only a root folder whose name matches the view. It is the only view type that has no *parent*, it forms the top of the project’s view hierarchy.

**Non-derived View** This is also called a *blank* view. Like the main view, it initially has no items except for a root folder. Although a non-derived view has a parent view, it is not *derived* from that parent, which means it does not inherit the parent view’s items. You can add new items to it, or you can share items from other views into it. A non-derived view can be used for non-lifecycle activities, acting like a “scratch pad” that you build up one item at a time.

<b>Derived View</b>	A derived view begins life as a subset or an exact copy of its parent view. One of the parent view's folders (often the root folder) is chosen as the root folder of the derived view; hence the derived view starts as a window into the artifacts at which it is rooted. What you can do with the artifacts in a derived view depend on whether it is a variant or reference view.
<b>Reference View</b>	A reference view is a derived view that is a pure subset of its parent view. A reference view does not have its own items; it uses the same items as its parent view. Consequently, updates made to a reference view (if allowed) are applied to the same items used by the parent view. This also means that reference views never have their own artifact branches; hence reference views are also called <i>non-branching views</i> . Reference views also do not have their own view labels, they share the same labels as their parents. Reference views are considered lightweight since they do not have their own items. A reference view can be updateable or read-only.
<b>Updateable Reference View</b>	An updateable reference view is a pure subset of its parent view with no added restrictions. If an item is updateable in the parent view, it is updateable in the reference view as well. Since the two views share the same items, changes in either view are immediately visible in both views. An updateable reference view is useful for exposing a portion of another view for security purposes. Instead of adding folder- and/or item-level security rights to the parent view, adding view-level security rights to an updateable reference child view is often easier to manage.
<b>Read-only Reference View</b>	A read-only reference view is a subset of items from the parent view that cannot be modified through the child view. A child view can <i>float</i> to the tip configuration of the parent view. In this case, it immediately reflects any changes made in the parent view to items it can see. Alternatively, a read-only reference view can be <i>pinned</i> to a specific configuration of the parent view: a timestamp, view label, or promotion state. A pinned read-only reference view reflects the state of the parent view (subset) as of that configuration. Once the child view is created, you can't change the configuration on which it is based. However, in the case of labels and promotion states, the child view will follow changes made to those objects. For example, if new items or different revisions are attached to the label or promotion state in the parent view, the child view will immediately reflect that change. Floating and pinned read-only reference views are useful when you want to subset a view and ensure read-only access, for example with applications such as build scripts.
<b>Variant View</b>	Unlike reference views, a variant view is not a pure subset of its parent view. Although a variant view may initially be created as an exact copy or subset of its parent view, it has its own items. In fact, when a variant view is first created, the parent view's items (or subset thereof) are shared (copied) to the child view, initially referencing the same artifacts. When new items are added to the child view, they may not be automatically added to the parent view depending on the containing folder's configuration. Furthermore, since it has its own items, the variant view's items may be independently configured, which means they could branch. Consequently, variant views are also called <i>branching views</i> . Whether or not a variant view's items are initially marked branch-on-change (BOC) is the major difference between the variant view subtypes.
<b>Branch-none View</b>	If a variant view's items are initially configured with BOC set to "false", it is referred to as a branch-none view. It acts a bit like a reference view, except that it doesn't share labels with its parent. If the child view's items are created with a pinned configuration, they will be read-only. If the item configurations float, updates through those items will float to the parent. This is why a branch-none view is sometimes called a "floating view". However, because the child view has its own items, "moves" in the child

will not propagate to the parent view. Because of the discrepancy between the propagation behavior of moves versus other updates, branch-none views can be very confusing. It is possible to create branch-none views and hand-tweak item configuration and branch-on-change, but this approach has proven tricky at best and sometimes disastrous. Consequently, we recommend against using branch-none views.

**Branch-all View**

Branch-all variant views are the most commonly-used views. A branch-all view begins life as a copy of its parent view (or subset thereof), with branch-on-change set to “true” for all items (that point to branchable artifact types). A branch-all view’s item can be configured to float, causing changes in the parent view to float to the child view. However, branch-all views are far more useful when all items are configured to a specific timestamp or view label.

**Lightweight activity view**

A Lightweight activity view is very similar to a branch-all variant view in that it can have its own items in addition to the parent view’s items. The main difference is that when a lightweight activity view is initially created it does not create copies of the items but rather uses the same items from the parent view. Items are shared down and branched on demand only when modified in the lightweight activity view. Note that only files are folders from the parent view are initially referenced in a lightweight activity view.

Of the six StarTeam view types, you will use main views and branch-all views with configured items (not floating items) the most. As detailed in the next section, each project will have only one main view, but you will use branch-all views to support both new development and maintenance activities. Reference views are occasionally used as a way to expose a read-only or updateable subset of its parent while simplifying security management. Blank views are rarely used, and branch-none views are not recommended.

## View Roles

Technically, you could manage a project’s artifacts using only the root view. For example, you could:

- Add all new folders, files, change requests, and other items directly in the root view.
- Acquire locks as you modify items to prevent conflicts between users.
- Employ revision labels and process items to identify revisions related to the same logical change.
- Create view labels to mark milestones such as specific builds and releases.
- Use promotion states to propagate snapshots represented by view labels through a coordinated test and release process.

When your project is first getting started, or if you have a very small (and safe) change to make, direct modification to the main view is fine. Otherwise, direct modification increases the risk of breaking the build if you haven’t staged your changes elsewhere first. The moment your team needs to work on two versions of the same file, module, or application at the same time, it needs a place other than the root view to do its work. This is the purpose of *development streams*. You need containers that support parallel development or maintenance activities that require different artifact branches within the project. The child views you create under the root view provide these containers.

Because version control and SCM products have existed for several decades, it should be no surprise that many different *patterns* have been developed for managing software development artifacts. These patterns affect the number of development streams you deploy and how you propagate changes between them. With StarTeam, this translates to the number and type of views you use and how they are organized.

Most of the child views in your view hierarchy should be branch-all views. To determine when you need a new view, where it should live in the hierarchy, and whether a different type of view would be more

appropriate, you should consider the role that each view will fulfill. Based on our experience, the best way to use views is to consider the roles described in the next sections.

## Main View: Home Base for Artifacts

The *main view*, also known as the *root view*, should contain the latest, approved revisions for the whole project. By latest, we mean that the main view should match your latest changes, ready for the next release. By approved, we mean that it should contain revisions that have undergone whatever verification checks your process requires: a complete build, unit testing, integration testing, etc. By whole project, we mean that it should not be a subset of the project's modules. In short, your main view should contain the latest, complete, production work, ready to be seen and used by everyone.

Implicit in this recommendation is that the main view should always be clean. That means it should be buildable and able to pass most if not all tests. Experience has shown that you will avoid many headaches by keeping your main view clean. To do this, all except for the simplest changes should be made in other views where they can be tested and fine-tuned until they're ready to be propagated to the main view. Once a new or modified revision has passed the point of no return—you're sure you won't change your mind—only then should it appear in the main view.

There is a subtle but important interaction between the main views and “share trees” you should be aware of. In general, the main view should contain the main (1.n) branch of each artifact. This happens automatically when you add a new artifact in the main view. StarTeam first creates a new 1.0 artifact and then connects it to the view and parent folder with a new item. But suppose you create a new artifact in a child view and then share the item “up” to the main view. Since the child view item was created first, it will be the “root share”, and the main view item will receive a “child share” item (initially pointing to the same artifact). In other words, the item share tree will point in the opposite direction as the views. You probably won't notice a problem until you modify the item in the main view, causing it to branch, thereby pointing to a non-main branch (e.g., 1.n.1.m). In future attempts to propagate changes between the two views, you'll find that it gets harder to propagate changes correctly because the share tree is backwards. For example, “rebase” operations won't work correctly. What can you do about this? The solution is to use the view compare/merge (VCM) facility to promote new items to the main view. VCM understands the share tree issue and propagates new items using an operation called *reverse share*.



**Note:** Savvy StarTeam pre-2006 users are aware of this subtle “share tree direction” issue and employ custom solutions. Some customers simply add new items to the root view first and then share them down to child views. Other customers propagate new child view items to a parent view by first moving them up and then sharing them back down, this is a basic version of what a VCM reverse share does.

## Activity View: Isolated Team Work Area

Suppose your team is assigned with developing a significant new enhancement (or a set of enhancements to be developed together). By significant, we mean changes that will affect more than a few artifacts, take more than a few days, and/or involve more than a few people. You wouldn't want to make changes directly to the main view since:

1. You want to check in your changes often to ensure they are backed-up, progress can be tracked, etc.
2. It may be a while before your changes are sufficiently stable to begin testing. The way to isolate your team's work is with an *activity view*.

An activity view is a branch-all view created from a well-identified, stable configuration of the main view. If the enhancement work only requires a portion of the main view's modules, you might choose to root the activity view from something other than the main view's root folder. Typically, an activity view is created from a view label or promotion state, which causes it to contain items pinned to the same revisions in the main view as of that snapshot. As changes are made to the activity view, the corresponding items will branch and hence will not be visible to the main view. New items are also added to the activity view and hence not visible to the main view. As work progresses, the state of tip revisions in the activity view may not always represent a buildable release. But eventually, your team will build, test, fix, and finish its work, whereupon it is “promoted” to the main view.

Starting 17.0 release of StarTeam server, "lightweight activity" views should be used for working on activity views. Creating a 'branch-all' view can be a heavyweight operation for large views as all the items are shared down during view creation and shares need to be fixed up. While a branch-all view is still recommended for release views which required to be built and labelled, lightweight activity views are recommended for quick feature development/ bug fixes. Creation time for lightweight activity view is extremely fast and the view itself leaves a low footprint on the server and database making it ideal for activity views.

Activity views typically have a limited lifespan: when the enhancement work is done and promoted, they can be deleted, usually after a certain period of time. Strictly speaking, a single activity view could be used for multiple enhancement activities, but there is an advantage to using separate views for each activity: if for some reason the activity must be cancelled, the activity view can be abandoned and eventually deleted. This isn't practical if the view contains work from multiple activities. Either way, activity views periodically require "rebasing". A variation of the activity view is an *integration view*, which supports integration activities for a large software project.

## Release View: For Post-Release Maintenance Work

When you release a snapshot of your software, you'll probably have to maintain a development stream just for hot fixes, patches, service packs, and so forth. A view that fills this role is called a *release view* (or a *maintenance view*).

Like an activity view, a release view is a branch-all view created from the main view as of a specific snapshot. A release view is created after one or more enhancement activities have been completed and promoted to the main view. It represents a milestone where your software has been (or is about to be) released externally. The release view is almost always rooted at the main view's root folder, and it is usually created from a frozen view label to clearly establish the software configuration that was actually released. In fact, many organizations create the release view first and then build and deliver the software from the release view.

Since it is a variant view, a release view can receive changes: for example, to fix bugs. If these bugs must be propagated to the main view, they are promoted as in activity views. However, release views are generally not rebased to receive changes from the main view except for bugs fixed in the main view first that must be applied to the release view as well.

If you need to make significant changes to a release view (perhaps a service pack), it is acceptable (even advisable) to create an activity view as a child of the release view. You would then make the changes in the activity view, perform appropriate validation tasks, and then promote it to the release view. You might even want to use a sandbox view, which is discussed below.

## Build View: Read-Only Windows for Build Scripts

Many StarTeam customers create build applications using simple build tools (for example: make, nmake), a commercial build product, or open source components (for example: Ant, CruiseControl). It is easy to integrate these tools with StarTeam due to the availability of its full-featured SDK and the availability of pre-built components such as StarTeam Ant tasks and a CruiseControl "bootstrapper" plugin for StarTeam. Build applications typically open a view, specify a snapshot configuration (timestamp, view label, or promotion state), and then check out the files they need.

In some cases, however, an organization may want to restrict the access of the build tool (or user) in comparison to the permissions other users have to the view. For example, you may want to guarantee that (a) the build tool can see only artifacts required by the build (and not design documents or other artifacts), and (b) the build tool has read-only access and cannot modify anything.

This situation could be handled through folder- or artifact-level security rights. However, this situation occurs often enough that some customers have found it useful to create a view tailored to the needs of the build tool. Such a *build view* is often created as a read-only reference view based on a promotion state. The reference view may be rooted at a non-root folder of the parent view. Consequently, a limited set of artifacts are exposed, which cannot be modified. Whenever the promotion state is assigned to a new view

label, the new artifact revisions automatically “appear” in the build view. That is, the build view follows changes to the promotion state.

Because security is easier to administer at the view level, a build view is often a more efficient way to accommodate build applications.

## Proper Use of Views

Regardless of view type, below are some general do’s and don’ts to consider when creating and managing views:

- Try to ensure that items in the main view refer to the main (1 . n) branches of its artifacts. Also, be careful not to delete the 1 . n branch of an artifact unless it has truly achieved end-of-life.
- *Leaf-most* views should be considered disposable. Eventually, when a view’s corresponding development activity is finished or its corresponding maintenance release is no longer supported, it should be deleted. This is important to prevent projects from growing unbounded, and it keeps the project structure uncluttered and easy to navigate. Deleting leaf views provides the **Server Administration** tool “purge” process with the maximum opportunity to actually shrink the database and vault by removing unneeded data.
- Three or four view levels in a project are normal. If you find your project has more than that, you might not be following best practices for views. For example, you may be making the mistake of performing new development work directly in the main view, forcing other active views to keep spawning child views. Or, you may be neglecting to promote the latest and greatest changes to the main view.
- Don’t use branch-none (“floating”) views except in extremely rare cases when you understand exactly how they work.
- You can “refactor” projects that have become too big by moving items in the main view to another project. Old change requests, tasks, and other projects can be moved to an “archive” project so they can still be accessed without cluttering the main view of an active project. If you have a project with numerous modules or applications, a large number of files can cause it to take too long to select “all descendants” in the main view. You can break the project up by creating new projects and moving (not sharing) folders and items corresponding to whole components or applications from the old project to the new projects. Do this when the main view reaches a major milestone such as after a new release. If you want to refactor the project by reorganizing the folder tree, do this before you split up the project. Create view labels before and after you refactor a project so you have markers for what changed.

## View Configuration Options

The following describes the view configuration options available in the **New View Wizard**.

<b>Floating Configuration</b>	Not recommended. All the items in the new view will be identical to the corresponding items in the current parent view. Changes to an item in the parent view will be made to the corresponding item in the new view until that item branches, while changes to an item in the new view will be reflected in the parent until the item in the new view branches. (In many cases, the first change to that item will result in branching). New items in the parent view will appear in a branching view. However, new items added to a branching view will appear in the parent view only if the new view is the <code>Branch none</code> type.
<b>Labeled Configuration</b>	All the items in the new view will have had the specified label in the parent view. In all cases, the revision of the item to which the label was attached is the tip revision in the new view. This option is disabled if the parent view has no view labels. Changes to the parent view do not affect the new view, including changes to the label upon which the view is based. Unless an item is set to <code>Branch on change</code> in the new view, it will be read-only and you cannot change it.






**Promotion State Configuration** All the items in the new view will have been part of the specified promotion state in the parent view. In all cases, the revision of the item that was part of the promotion state is the tip revision in the new view. This option is disabled if the parent view has no promotion states defined for it. Changes to the parent view will not affect the new view, including changes to the promotion state or its assigned label. Unless a specific item is set to `Branch on change` in the new reference view, it will be read-only and you cannot change it.


**Configuration As Of** The new view will contain only the items that existed at the date and time you specify. In all cases, the tip revision of each item in the new view is the revision closest to, but before, the specified time. Changes to the parent view will not affect the new view. Unless a specific item is set to `Branch on change` in the new view, it will be read-only and you cannot change it.

## View Type Options and Settings

The table below lists the settings that must be selected in the **New View Wizard** to create the different views.

Desired Characteristics for New View		Options to Set	
<b>Branching: Branch All (Not Floating)</b>		<b>View Type</b>	<code>Branch All</code> .
<b>New Items</b>	New items in the child view do not appear in the parent view. New items in the parent view do not appear in the child view.	<b>Root Folder</b>	Selected from parent view.
<b>Existing Items</b>	Existing items in the child view are the same as in parent view at the time of configuration, until the item in the child view branches. Changes cannot be made to an item in the child view if the change does not result in branching.	<b>Working Folder</b>	Should be different from that of the parent, to avoid conflicts.
<b>Item Behavior</b>	<b>Branch on Change</b> check box is enabled and selected for all items that can branch. Any change to a child item that can branch results in the branching of that item, unless its <b>Branch on Change</b> check box has been cleared. After branching, the check box is disabled.	<b>Configuration</b>	Other than floating ( <code>Labeled</code> , <code>Promotion State</code> , or <code>As of specific date</code> ).
			<b>Note: Addressed In Build</b> field when the child view is created, <code>Next Build</code> will not be replaced by a build label until the change request branches.
<b>Read-only Reference View (Frozen)</b>		<b>View Type</b>	<code>Read-only Reference</code> .
<b>New Items</b>	New items cannot be added to child view. New items in parent view appear in child view.	<b>Root Folder</b>	Selected from parent view.
<b>Existing Items</b>	Existing items are the same in the child view and the parent view; they can be changed only from the parent view.	<b>Working Folder</b>	Usually the same as that of the parent view.
<b>Item Behavior</b>	<b>Branch on Change</b> has the same setting as the parent item, but is irrelevant; no change can be made.	<b>Configuration</b>	Other than floating ( <code>Labeled</code> , <code>Promotion State</code> , or <code>As of specific date</code> ).
		These views can be rolled back.	
<b>Non-derived View (also called Blank Branching View)</b>		<b>View Type</b>	<code>Non-Derived</code> .
<b>New Items</b>	New items in child view do not appear in the parent view. New items in the	<b>Root Folder</b>	N/A.

Desired Characteristics for New View		Options to Set	
	parent view do no appear in the child view.	<b>Working Folder</b>	Should be different from that of the parent to avoid conflicts.
<b>Existing Items</b>	Existing items in the parent view do not appear in the child view.	<b>Configuration</b>	N/A.
<b>Item Behavior</b>	<b>Branch on Change</b> check box disabled.		
<b>Reference View</b> (Also called <b>Read/Write Reference View</b> )		<b>View Type</b>	Reference.
<b>New Items</b>	New items in the child view appear in both views. New items in the parent view appear in both views if they are in the subset accessed by the child view.	<b>Root Folder</b>	Selected from parent view.
<b>Existing Items</b>	Existing items are the same in the child view and the parent view. They can be changed from either view.	<b>Working Folder</b>	Usually the same as that of the parent view.
<b>Item Behavior</b>	<b>Branch on Change</b> has the same setting as the parent item.	<b>Configuration</b>	N/A. Always floats.
	<b>Note:</b> Labels created and objects deleted in the child view appear and disappear in the parent view; this is not true for other types of child views.		
<b>Lightweight Activity View</b>		<b>View Type</b>	Branch-all.
<b>New Items</b>	New items in the child view do not appear in the parent view. New items in the parent view do not appear in the child view.	<b>Root Folder</b>	Parent view root folder.
<b>Existing Items</b>	Existing items in the child view are the same as in parent view at the time of configuration, until the item in the child view branches. Changes cannot be made to an item in the child view if the change does not result in branching.	<b>Working Folder</b>	Should be different from that of parent to avoid conflict.
<b>Item Behavior</b>	Any change to a child item results in the branching of the item. Item behavior cannot be modified.	<b>Configuration</b>	As of specific date.
	<b>Note:</b> Labels created and objects deleted in the child view appear and disappear in the parent view; this is not true for other types of child views.		
Branching: <b>Branch None (Not Floating)</b> . Not recommended.		<b>View Type</b>	Branch None (an advanced type).
<b>New Items</b>	New items in the child view do not appear in the parent view; new items in the parent view do not appear in the child view.	<b>Root Folder</b>	Selected from parent view.
<b>Existing Items</b>	Existing items in the child view are the same as in the parent view at the time of configuration, until the item in the child view branches. Changes cannot be	<b>Working Folder</b>	Should be different from that of the parent, to avoid conflicts Configuration.
		<b>Configuration</b>	Other than floating (Labeled, Promotion State, or As of specific date).

Desired Characteristics for New View		Options to Set
	made to an item in the child view if that change does not result in branching.	 <b>Note: Addressed In Build</b> field when the child view is created, <code>Next Build</code> will not be replaced by a build label until the change request branches.
<b>Item Behavior</b>	<b>Branch on Change</b> check box enabled, but initially cleared.	
No change to a child item that can branch results in branching until the Branch on Change check box is selected. After branching, the box is disabled.		
Branching: <b>Branch None (Floating)</b> . Not recommended. Use only when a different set of view labels is needed for the same data.		<b>View Type</b> Branch None (an advanced type).
<b>New Items</b>	New items in the child view appear in the parent view; new items in the parent view appear in the child view if they are in the subset accessed by the child view. In the child view, new items from the parent have the <b>Branch on Change</b> check box cleared.	<b>Root Folder</b> Selected from parent view.
<b>Existing Items</b>	Existing items are the same in the child view as in the parent view; they can be changed in either the parent or child view until the item in the child view branches. However, items deleted from one view are not deleted from the other.	<b>Working Folder</b> Should be different from that of the parent to avoid conflicts.
<b>Item Behavior</b>	<b>Branch on Change</b> check box enabled, but initially cleared	<b>Configuration</b> Floating. See the note below.
No change to a child item that can branch results in branching until the Branch on Change check box is selected. After branching, the check box is disabled.		
Branching: <b>Branch All (Floating)</b> . Not recommended.		<b>View Type</b> Branch All, Float (an advanced type).
<b>New Items</b>	New items in the child view do not appear in the parent view; new items in the parent view appear in both views if they are in the subset accessed by the child view. In the child view, new items from the parent have the <b>Branch on Change</b> check box selected.	<b>Root Folder</b> Selected from parent view.
<b>Existing Items</b>	Changes to existing items in the parent view appear in the child view until the corresponding item in the child view branches. Changes to existing items in the child view can appear in the parent view, but only if the <b>Branch on Change</b> check box for that item is cleared. However, items deleted from one view are not deleted from the other.	<b>Working Folder</b> Should be different from that of the parent to avoid conflicts.
<b>Item Behavior</b>	<b>Branch on Change</b> check box is enabled and initially selected for all items that can branch.	<b>Configuration</b> Floating See the note below.
Any change to a child item that can branch results in the branching of that item, unless its Branch on Change		

Desired Characteristics for New View		Options to Set	
check box has been cleared. After branching, the box is disabled.			
Branching: <b>Branch All (Floating)</b>		<b>View Type</b>	Branch All.
<b>New Items</b>	New items in the child view do not appear in the parent view. New items in the parent view appear in both views if they are in the subset accessed by the child view. In the child view, new items from the parent have the <b>Branch on Change</b> check box selected.	<b>Root Folder</b>	Selected from parent view.
		<b>Working Folder</b>	Should be different from that of the parent to avoid conflicts.
		<b>Configuration</b>	Floating
		See the note below.	
<b>Existing Items</b>	Changes to existing items in the parent view appear in the child view until the corresponding item in the child view branches. Changes to existing items in the child view can appear in the parent view, but only if the <b>Branch on Change</b> check box for that item is cleared. However, items deleted from one view are not deleted from the other.		
<b>Item Behavior</b>	<b>Branch on Change</b> check box is enabled and selected for all items that can branch.		
Any change to a child item that can branch results in the branching of that item, unless its <b>Branch on Change</b> check box has been cleared. After branching, the box is disabled.			



**Note:** If users are likely to perform many move and share operations, using branching, floating views can result in multiple unwanted references to the same folders or items, causing confusion. Also, if a change request has `Next Build` in the **Addressed In Build** field when the child view is created, `Next Build` will be replaced by the parent's next build label, unless the change request is first branched.

## Working with Views

This section contains tasks related to using and managing views.

### Opening Perspectives and Views

1. Choose **Window > Perspective > Open Perspective > Other** from the main menu. The **Open Perspective** dialog box opens.
2. Select either the StarTeam Activity or StarTeam Classic perspective.
3. Click **OK**.

The specified perspective opens.

### Creating and Configuring Views

This topic provides the basic procedure for creating a new view based upon an existing view.

1. Display the project view upon which the new view will be based.
2. Choose **StarTeam > View > New**. The **New View Wizard** opens.
3. Select one of the available options from the **View Type** list:

<b>Branch All</b>	Will be based on a configuration of the currently open view. All items in the new view will be set to branch when they are modified. This branch behavior can be changed later for individual items in the new view so that changed items do not branch.
<b>Reference</b>	Allows users to read from and write to a subset of the parent view's current configuration. Any changes appear in both the reference view and its parent.
<b>LightWeight Activity</b>	View will be based on the current configuration of the currently open view. All file and folder items in the new view will be shared down on demand and set to branch when they are modified. Project should have "Create workspace change package" option enforced to be able to create lightweight activity views. Skip steps 5 to 9.
<b>Read-only Reference</b>	Allows users to read from a subset of the parent view. Unlike a read-write reference view, the contents of a read-only reference view may be current (floating) or configured to a point in the parent view's past by specifying a label, promotion state, or time.

4. Type a **Name** and a **Description** for the view in the appropriate fields and click **Next**.
5. Select the **Root Folder** for the new view and click **Next**.



**Note:** The **New View Wizard** skips this step for a **Non-Derived** view.

6. Type or browse for the name of an appropriate **Default Working Folder**.



**Caution:** For a **Branch All** or a **Non-Derived** view, always use a working folder that is different from the one used by the parent view. Using the same working folder for the parent and child views can cause changes in one view to be overwritten when files are checked out from the other view. It can also result in incorrect or, at least, misleading file status indicators. For a **Reference** or **Read-Only Reference** view, you can use the same working folder as the parent view.

7. Click **Next** to display the **Select Types** page.
8. Select the item types to include in the new view. The new view will include items of the selected types from the parent view, unless the **Override default types** option has been selected in the **View Access Rights** dialog box.



**Tip:** To create a view with no shared items, use the **Branch All** view type and clear all the check boxes on the **Select Types** page.

9. Click **Next** to display the **Configuration** page.

If you are creating a **Non-derived** or **Reference** view, click **Finish**.



**Note:** It is not necessary to display the **Configuration** page for a **Non-Derived** view because no items from the parent view are included. It is also not necessary to display the **Configuration** page for a **Reference** view because the items have the same configuration as those in the parent view.

10. Select one of the available configuration options on the **Configuration** page.
11. Click **Finish**.

## Lightweight Activity View Restrictions

As explained previously, a "lightweight activity view" type is best useful as activity views for developer and is generally preferred when the views are large and a branch-all view creation is costly. However it is important to note there are number of restrictions in a lightweight activity view compared to a regular branch-all view which have to be taken into consideration before creating the view. The following operations are not permitted in a lightweight activity view:

- Creation of labels and promotion states.
- Lock/unlocking of files/folders.
- Setting item level and container level access rights and clone access rights.

- Modifying Item behavior.
- Create links between items.
- Whilst VCM promotes and rebase from/to a lightweight activity view are allowed, VCM replicate from a sibling view to a lightweight activity view is disabled.
- Opening a committed change package in a lightweight activity view.
- Cross view sharing of a lightweight item.
- Lightweight activity views can only be rolled back as of a configuration since labels/promotion states are disabled.
- Item level Access rights cannot be modified inside a lightweight activity view.

## Deleting Views

Before deleting a project view, be absolutely certain that you wish to do so. After performing this operation, you will no longer be able to access any item in the view that is not shared with another project or view. Deleted views are also not visible in the **Select View** dialog box, although deleting a view does not remove any data from the server database.

If other users are connected to the project view when it is deleted, they will receive a Deleted message the next time they initiate a view command.



**Note:** A view cannot be deleted if it has derived child views.

1. Open the Server Explorer.
2. Right-click on the view name, and choose **Delete**.
3. Click **Yes**. A confirmation dialog box asks you to type the name of the project.
4. Type the project name, which is case-sensitive, in the **View Name** field.
5. Click **OK**.

## Reviewing or Modifying View Properties

Sometimes you may want to look at the values and properties originally used to create a view. Reviewing this information may help you understand the behavior of changes within the view or the views that have been derived from it. Also, if you have access rights to do so, you may be able to modify view properties as well.

1. Choose **StarTeam > View > Properties** . The View **Properties** dialog box opens.
2. Select **View**.
3. Select the **Name** tab to see or change the following:
  - Name and description of the view
  - Who created the view and when it was created (read-only)
  - Whether the items are set to branch on change
  - Whether a central or a per folder repository is being used
  - Working folder path



**Note:** Your access rights determine which items you can change.

4. Select the **Hierarchy** tab to review the list of views for this project and their relationship to one another.
5. Select the **Type** tab to see:
  - View type.
  - Whether the view is a root, branching, non-derived, or a reference view.
  - For branching views, whether the original default was Branch All or Branch None.
  - Parent view on which this view was based.
  - Parent configuration used to create this view.

6. Click **OK**.

## Creating View Labels

View labels can be extremely useful when you want to label every folder and item in a particular view.

1. Do one of the following:

- With a StarTeam perspective or view open, choose **StarTeam > View > Labels** or **StarTeam > View > Properties** from the main menu.
- In the Server Explorer, right click on the view node, and choose **Labels** or **Properties** from the context menu.
- In the Navigator or Package Explorers, right click on the project, and choose **Properties** or **Team > Labels**.

The **Properties** dialog box opens.

2. In the resulting dialog box, do one of the following:

- If opened from the Navigator or Package Explorers, expand **StarTeam Provider** and select **Labels (for view)**.
- If opened from the StarTeam main menu or Server Explorer, select **Labels**.

This **View** tab lists existing view labels in reverse chronological order, based on the time at which they were created.

3. Click **New** to create a new label and add its name to the list box. The **View Label** dialog box opens.

4. Type a name and description for the label in the appropriate text boxes.

The maximum label name length is 64 characters, and the description length is 254 characters.

5. Optional: Check **Use As Build Label** to update each change request that has **Next Build** as the setting for its **Addressed In Build** property. If this option is not selected, change requests will still have the view label, but it will not affect the setting of the **Addressed In Build** property.

6. Optional: To freeze the label so that the revisions attached to it cannot be changed, check **Frozen**.

7. Select one of the following options:

**Current Configuration** Attaches the label to the tip revision.

**Labeled Configuration** Attaches the label to the revision with a specified label. The labels are in reverse chronological order based on the time at which they were created.

**Promotion State Configuration** Attaches the label to the revision currently in a specified promotion state. (Actually, the label is attached to the revision that has the promotion state's current view label.)

**Configuration As Of** Attach the label to the revision that was the tip revision at a specified date and time.

8. Click **OK**.





**Note:** If you select the current time for the label configuration, it is important to synchronize the dates and times of the computers that run the clients and the server. Otherwise, the labels may not be immediately visible.

## Changing a View's Default and Alternate Working Folders


Make sure that everyone is logged off from the server and that the server is locked before you change the *Default Working Folder*. It is just as critical to perform these actions as it is when you change custom fields or do anything else that affects all users.

When you change the **Default Working Folder**, not only the path to the working folder but the path to each child folder in the view may be similarly modified—not just for you, but for everyone working with that view.

 **Caution:** Do not change the **Default Working Folder** unless you are a project administrator. These default settings affect all users and incorrect settings cause other users to be unable to check out StarTeam files. The default settings should only be set to the name of the folder. If you want to use a different location for your working folder than the **Default Working Folder** path, specify an **Alternate Working Folder** path.

 **Important:** Changing the view's working folder for working Eclipse projects is not recommended. Eclipse projects should be recreated after changing the view's working folder. If you want to use a different location for your working folder other than the Default Working Folder path, it is recommended that you specify an Alternate Working Folder path.

1. Choose **StarTeam > View > Properties** to open the **Properties** dialog box.
2. Select **View**.
3. Click the **Name** tab.
4. Do one of the following:
  - Select **Alternate** to create a different working folder for only yourself.
  - If you are a project administrator, select **Default** to specify the default repository path for all users.
5. Type the name of a new working folder or browse for a path to a working folder. If you browse for the path, it becomes an absolute path. This path can be edited, however, to enable you to work on a computer that uses a different letter for its hard drive.

 **Note:** It is important that the **Default Working Folder** point to a location that is physically discrete for each user, such as a drive on that user's workstation or a personal directory on a shared file server.


## Switching Between StarTeam Project Views

By default, the newly selected view always opens with the Current configuration, regardless of the configuration it had when you last exited it.

1. In the Package or Navigator Explorers, right-click on the root project node, and choose **Properties**. The **Properties** dialog box opens.
2. Select **StarTeam Provider** from the menu tree.
3. Click **Change View**. A dialog box opens showing all views for the project.
4. Select the desired view and click **OK**.
5. Click **OK** to close the **Properties** dialog box.

## Copying View Labels

Occasionally, you may want to create a view label and attach it to the same item revisions as an existing view label, with a few additions or exceptions. The steps in this procedure explain how to create a view label based on an existing view label. For example, suppose builds are done only after a view has been rolled back to a label and that the build is given the same name as the label. If, in the last build, only one Help file was missing, you would probably change the existing label to include that one file and rebuild. However, if the previous build was already made available to users participating in a field test, using the same label could cause confusion. It would be better to create a new view label as a copy of the older label and then add the missing file to the new label.

 **Note:** You cannot copy a view label unless it already exists in the view in which you are performing this operation. The view configuration must also be current.

1. Choose **StarTeam > View > Labels** .
2. The **Properties** dialog box opens. Select **Labels (for view)**.
3. Click **New**. The **View Label** dialog box opens.
4. Type a name and description for the label. The maximum name length is 64 characters and the description length is 254 characters.



5. Select the **Labeled Configuration** option to attach the label to item revisions that have an existing label.
6. Optionally, uncheck **Use As Build Label** if you do not want this label to be a build label.



**Note:** By default all view labels are designated as build labels.

7. Click **OK**.
8. Click **Close**. The new view label is now attached to the same revisions as the existing label. You can now use the context menus provided in the Eclipse Explorers or StarTeam views to attach or detach labels for which the new label must differ.
9. Select the items in the Package or Navigator Explorers, and choose **Team > Labels > Detach** from the context menu.
10. Attach the new label to items formerly not included, and/or attach the new label to different revisions of items to which it is already attached. Select the items in the Package or Navigator Explorers, and choose **Team > Labels > Attach** from the context menu.

## Promoting View Labels

You can promote a view label from one promotion state to the next if you have the appropriate access rights, and when it satisfies the criteria established for that state.

1. Choose **StarTeam > View > Promotion** or **StarTeam > View > Properties**

*Alternative 1:* In Server Explorer, right click on the view node, and choose **Promotion** or **Properties**.

*Alternative 2:* In the Navigator or Package Explorers, right click on the project, and choose **Properties** or **Team > Labels**.

The **Properties** dialog box opens.

2. In the resulting dialog box, do one of the following:

- If opened from the Navigator or Package Explorers, expand **StarTeam Provider** and select **Promotion**.
- If opened from the StarTeam main menu or StarTeam, select **Promotion**.

The states are displayed from the final state down to the initial state.

3. Select the promotion state currently associated with the view label that you want to promote.
4. Click **Promote**. The dialog box indicates that the view label is now associated with the next state (the state immediately above the selected state in the **Properties** dialog box).
5. Click **OK**.

The view label now applies to both the original state and the new state. Usually, your next action is to associate a new view label with the original state.

## Demoting View Labels

1. Do one of the following:

- With a StarTeam perspective or view open, choose **StarTeam > View > Promotion** or **StarTeam > View > Properties**.
- In the Server Explorer, right click on the view node, and choose **Promotion** or **Properties**.
- In the Navigator or Package Explorers, right click on the project, and choose **Properties** or **Team > Labels**.

The **Properties** dialog box opens.

2. In the resulting dialog box, do one of the following:

- If opened from the Navigator or Package Explorers, choose **StarTeam Provider > Promotion**.
- If opened from the StarTeam main menu or Server Explorer, select **Promotion**.

3. Click **Edit** to open the **Promotion State** dialog box.
4. Select a different view label from the **View Label** list box.
5. Click **OK**.

## Branching

A branching view is a view that permits branching. This means that the folders and other items in the view can separate from the corresponding items in the parent.

Branching views serve many purposes. For example, you can create a branching view to:

- Meet different needs from those of your main line of development. For example, you might create a maintenance release or a custom version of your product, branched from a prior commercial release.
- Start development on the next release of your product by using some or all of the files from the previous release.
- Keep an area of your project private until it is completed and tested. Then you can merge your changes into the main line of development when and where necessary.

Only folders, files, and change requests can branch, although not every folder or every item in a branching view must branch. Requirements, tasks, and topics never branch.

Until an item branches, the corresponding items in both views remain identical. After an item branches, they are no longer identical, and the revision number indicates the new branch. The only way to make the items identical again is to manually merge them by comparing and merging views. After branching occurs, StarTeam no longer sends updates to nor applies updates from the corresponding item in the parent view.

For reasons of safety, deletions made in the parent view are not propagated to the child view and vice versa. If you want to delete a folder or item from all related views, you have to delete it manually from each of those views.

Also, a move is considered a copy operation followed by a delete operation. Consequently, the view in which the move was made has one copy of a folder or item in the new location, while the related views have two copies of the folder or item, one in the original location and one in the new location.



**Note:** Branching a view negates all shares, not just the ones between parent and child views.

## Branching Options

Branching occurs when an item in the child view changes if its behavior is set to **Branch On Change**. When an item branches, a separation occurs between the item and its corresponding item in the parent view. These separate items also begin to have different branch revision numbers.

When creating a branching view, if you select:

**Branch All** The behavior of every item that is in the view at the time the view is created is set to **Branch On Change**.

**Branch None** The behavior of every item in the view at the time the view is created is not set to **Branch On Change**. Changes to any item with a floating configuration can be propagated to the parent view.

When you branch a view, any manual shares between items in the same view are not retained in the view's child view.



**Note:** Any item with a frozen or fixed configuration is read-only when its behavior is not set to **Branch On Change**. Read-only means that no data about this item within the view can be changed. For example, although you may be able to edit a file, you cannot check it in or change its properties.

As you add, move, share, and modify items, their behaviors can change.

### **Branching is Disabled**

When the check box for **Branch On Change** is disabled the item cannot branch. One of the following is true:

- The item is original to the current view, not shared into it. In other words, it is the root item in its own reference tree.
- The item has already branched. (An item can branch only once per view.)

### **Branching Is Set to Branch On Change**

When **Branch On Change** is both enabled and selected, branching occurs the next time the item changes. At that time, a separation occurs between the item in the new view and its corresponding item in the parent view. The item that becomes separated from its corresponding item in the parent view takes on the following behaviors:

- Its **Branch On Change** check box becomes disabled.
- Its revision number's dot notation expands to include two more numbers.

### **Branching Is Not Set to Branch On Change**

When the **Branch On Change** check box is enabled but cleared, branching does not occur when you change the item.

If the item's configuration floats, the change is propagated to the parent view.

If the item's configuration does not float, the item cannot be changed because the parent view cannot be updated. The item is treated as though it were read-only. For example, if the item is a file, you can edit it but you cannot check it in or change its properties.

## **Branching Behavior of Items**

Given the appropriate settings for folders, files, and change requests, you can branch these items in a child view—that is, you can separate these items from the corresponding items in the parent view.

Branching a folder does not branch its contents (child folders nor items.)

After an item branches, it receives a new revision number. For example, if a file's revision number (in dot notation) 1.13 before the file branches, it becomes 1.13.1.0 after branching. The next change to the file in the parent view will receive the revision number 1.14. The next change in the child becomes 1.13.1.1.

Below are the basic facts about branching behavior:

- Folders and change requests branch when their properties change.
- Files branch when either their contents or their properties change.
- Requirements, tasks, and topics can never branch.

### **Typical Branching Scenario**

Suppose you are working on a product and a customer requests a special edition of the product with a few special features tailored specifically for that customer. To separate the current product's items from those for the special request, a branching view is created.

When items are branched, they are derived from other items that become their ancestors. Items may have several completely different revision histories with common ancestries. In the case of a text file, for example, the branched item can later be merged with the file from which it originated. For example, the development of a product for a new operating system may start with the existing files for the first operating system as its base.

### **History Affects Branching Behavior**

Whether or not a folder, file, or change request has the ability to branch depends on its history. If you do not know the complete history, you should not assume that you know its behavior. For example,

- If a folder or item was in the parent view at the time the branching view was created, and if the branching view was created with **Branch All** as its branching option in the **New View Wizard**, the folder

or item's branching behavior is initially enabled and the **Branch On Change** check box is selected in the **Folder Behavior** dialog box.

- If a folder or item was in the parent view at the time the branching view was created, and if the branching view was created with **Branch None** as its branching option, the folder or item's branching behavior is initially enabled and the **Branch On Change** check box is cleared. However, this behavior can be changed.
- If a folder or item is added to the branching view after the view is created, the folder or item's branching behavior is disabled. The **Branch On Change** check box is disabled and cleared in the **Folder Behavior** dialog box. However, if you share that folder or item, its branching behavior becomes enabled automatically in its new view.

### Branching Behavior of Shared Items

The "branch on change" behavior of a shared item is specific to the folder it is in and the "branch on change" check box is selected by default for the shared file.

## Effects on Change Requests When Branched, Moved, and Shared

The workflow of a change request may be significantly affected when the change request is moved, merged, or branches:

- If the **Last Build Tested** and the **Addressed In Build** fields have build labels as their values (if these fields are not empty and do not contain the value **Next Build**) the altered change request retains those values. In the new view, these values can be changed, but only to the names of build labels that exist in that view.
- If the **Addressed In Build** field contains the value **Next Build** at the time of the operation, this value is replaced by the name of the next build label created in the original view, not the next build label created in the new view. This action occurs even if other alterations have been made to the change request in the new view.
- If the **Last Build Tested** and the **Addressed In Build** fields have no values at the time of the operation, their workflow is specific to the view in which they currently reside.



**Note:** If a change request branches, its workflow is affected by its values in the **Last Build Tested** and the **Addressed In Build** fields at the time it branches.

## References Overview

You can base a folder or item in one application location on another folder or item stored in a different location within the same server configuration. **References** indicate the relationships between an original folder or item and the others based on it. References can be used to decide whether the changes you have made to a folder or item in one location need to be applied elsewhere.

Found in the lower pane of the clients, the **Reference tab** shows the relationships between the selected item and other folders or items with which it is associated. A folder or item may be associated with more than one project, view, or parent folder in the same server configuration because of sharing or because a child view has been created. Each instance of the original folder or item has a reference. Item references (including folders) can be viewed on the Reference tab of the lower pane. You can also view folder references from the **Folder** tree in the left pane by selecting **Advanced > References** from the **Folder** tree or context menu to display a dialog.

## Understanding References

StarTeam creates at least one reference to a folder or item whenever:


- You create or add a folder or item.

- A branching child view is created that will contain that folder or item. As a branching child view is created from its parent, a subset of the folders or items in the parent becomes part of the child view. StarTeam automatically shares the folders and items in that subset into the child view.
- You manually share a folder or item from one location to another.

As you add, share, or move a folder or item, more than one reference to it may occur if the view is a child view that branches and floats, or if the view has child views that branch and float.

### Actions Causing StarTeam to Create References

For example, suppose you want to move a file from one folder to another in the same view. Suppose that the view has two child views, both of which contain the file. That means that there are at least three references to this file, one in each of three views. Now, you move the file to another folder in the same view. The reference in the current view is moved to represent the new location of the file. Depending on the properties of the two child views, a new reference may be created for the file in each of the child views. The references in those child views to the file in its original location still exist, because the application does not assume that you want to change those references just because you have moved the file in the current view. You may end up with five references to this file that formerly had three references.

 **Note:** Most administrators avoid branching, floating views if users are likely to perform many operations that result in additional references. For example, moving and sharing can result in multiple unwanted references to the same folders or items, which can cause confusion.





The following table explains what references StarTeam creates in the current view, the recipient view, the parent of the recipient view, and the children of the recipient view. This is often recursive. For example, if a reference is created in the parent view, new references might be created in the other children of that view or in the parent of that view, and so on, depending on what views are floating.

When a folder or item is...	...is a reference added to the view of the recipient?	...is a reference added to the parent view of the recipient view?	...is a reference added to the child views of the recipient view?
Part of a newly-created view	Yes, unless the new view is a reference view. In this case, a new view is not really being created, because a reference view is just a new way of looking at an existing view.) There is one reference for the folder or item in the newly-created view.	No, because the parent view is the source of the folder or item, so the reference in the parent view already exists.	No, because the newly-created view has no child views.
Added to the current view	Yes, there is one reference for the new folder or item in the current view.	Yes, if the current view is a branch none, floating child of the parent view.  Otherwise, no.	Yes, if the child view is a branching (either branch none or branch all), floating child of the current view.  Otherwise, no.
Shared within the current view	Yes, a new reference is created for the shared folder or item in the new location in the current view.	Yes, if the current view is a branch none, floating child of the parent view.  Otherwise, no.	Yes, if the child view is a branching (either branch none or branch all), floating child of the current view.  Otherwise, no.
Moved within the current view	No, the original reference is updated to reflect the move.	Yes, if the current view is a branch none, floating child of the parent view.	Yes, if the child view is a branching (either branch


When a folder or item is...	...is a reference added to the view of the recipient?	...is a reference added to the parent view of the recipient view?	...is a reference added to the child views of the recipient view?
		Otherwise, no.	none or branch all), floating child of the current view.
			Otherwise, no.

## How to View References

Consider the following example of four references that would display in the **Folder References** dialog box:

-  Help Files::Help Files::Help Files::starteam,1.0
-  Help Files::Help Files\Freeze Check::Help Files\release 4\starteam, 1.0
-  Help Files::Help Files\Freeze Check\New View::\Help Files\release 4\starteam, 1.0
-  Help Files::Help files\variant 2::Help Files\release 4\starteam, 1.0.1.2

In the above example, the selected folder has four references.

The Current icon  indicates which reference represents the currently selected folder or item. Otherwise, this dialog contains the same information regardless of the view in which you selected the folder.




Each reference shows the following, separated by double colons (::):




- The project name (for example, Help Files).
- The path from the root view to the view containing the folder (or item). For example, Help Files\Freeze Check\New View, where Help Files is the name of the root view, Freeze Check is a child of the root view, and New View is a child of the Freeze Check view.
- The path to the folder within the view. In the case of an item, the path is to the parent folder of the item.
- In the case of an item, the name or number associated with that item. This can be the filename, change request number, the requirement number, the task number, or topic number.
- The tip revision number for the folder (or item) in that view. (This information is separated from the rest of the reference by a comma, rather than the double colon.) For example, the folder in the example is revision 1.0 in all views except for the variant 2 view (see the last leaf in the example tree). In the variant 2 view, the revision number for the folder is 1.0.1.2 which indicates that the folder has been branched from the 1.0 revision in its parent view and has had three revisions in the variant 2 view. Those revisions are 1.0.1.0, 1.0.1.1, and 1.0.1.2.

In this example, the name of the project, the name of the root view, and the root folder in the root view all have the same name.

You can resize the **Folder References** dialog box (by dragging an edge or corner). It displays scroll bars when appropriate. The references in bold indicate which revisions of the currently selected folder or item are its descendants. In other words, the currently selected folder or item is part of the revision history for the references that are in bold.

Consider the following example from the Reference tab shows the references for a file (AUDITSCC.DOC). The reference for the currently selected file indicates that revision of the file is 1.6. As indicated by the bolding of its reference, revision 1.8 is the only descendant of revision 1.6. If a defect is found in revision 1.6 of AUDITSCC.DOC, the bolding helps you determine which descendants of 1.6 may also need the corrected lines. In this case, you may only new to update 1.8.

-   Help Files::Help Files::Help Files\starteam::AUDITSCC.DOC, 1.8
-  Help Files::Help Files\Freeze Check::Help Files \starteam::AUDITSCC.DOC, 1.1

-  Help Files::Help Files\Freeze Check::New View2::starteamp::AUDITSCC.DOC, 1.1.1.0
-  Help Files::Help Files\varc::Help Files\starteamp::AUDITSCC.DOC, 1.6
-  Help Files::Help Files\variant 2::starteamp::AUDITSCC.DOC, 1.2

## Initial References

When you add a folder or item to the application, StarTeam creates a reference. Consider the following example of a folder hierarchy for a newly-created project. In this example, it is the folder hierarchy for the root view of that project.

Before you make any changes to the folder properties of the `Source Code` folder, the **Folder References** dialog box would contain exactly one reference to it. For example, `Big Product::Big Product::Big Product\Source Code, 1.0`.

As you make changes to the folder properties of the `Source Code` folder, the revision number might change from 1.0 to 1.1 and later 1.2. However, there will still be only one reference to this folder in the **Folder Reference** dialog box.

If a reference view is created (to be used, for example, by a group of reviewers), the view hierarchy for the `Big Product` project would contain two views, but the `Source Code` folder would continue to have just one reference. A Reference view contains a subset of the folders in its parent view, but those folders are the same folders as those in the parent view. They cannot branch. For example, after creating a reference view for reviewers, the Folder References dialog box would contain the following information:

### **Big Product::Big Product::Big Product\Source Code, 1.2.**

Once you share the folder manually or share it automatically when you create a branching child view, additional references then display in the **Folder References** dialog box.



**Tip:** Do not confuse reference views with folder and item references. A reference view looks like a new view, but it is really a subset of an existing view. A folder or item reference is like a reference count. It indicates how many copies of the object exist or can exist if the object branches in each of its new locations. The creation of a reference view does not result in the creation of any folder or item references.

## References Created by Branching Views


When you create a branching view, each folder or item automatically shared from the parent view to the child view acquires an additional reference. In the view hierarchy (which you can display from **View > Select View**), the new reference is a child of the original reference.


### **Folder References Created by Branching Views**

Suppose that when the 1.0 version of `Big Product` ships, the team leader creates a branching view (based on the ship date for the 1.0 version) to be used for service packs, while new development on version 2.0 still continues in the project root view. These actions would result in the following view hierarchy:



- `Big Product`
- `Big Product 1.0 Plus Service Packs`
- `Reference view for reviewers`



At this point, two references display in the **Folder References** dialog box. When you are in the root view, `Big Product`, the Folder References dialog box for the `Source Code` folder contains the following information:

-  `Big Product::Big Product::Big Product\Source Code, 1.2`



-  Big Product::Big Product\Big Product 1.0 Plus Service Packs::Big Product\Source Code, 1.2

When you are in the child view, Big Product 1.0 Plus Service Packs, the **Folder References** dialog box for the Source Code folder contains the following information:



-  Big Product::Big Product::Big Product\Source Code, 1.2
-  Big Product::Big Product\Big Product 1.0 Plus Service Packs::Big Product\Source Code, 1.2

The Current (You Are Here)  icon indicates which reference represents the currently selected folder or item. Otherwise, this dialog contains the same information regardless of the view in which you selected the folder. StarTeam indents the reference for a child view beneath the reference for its parent. The references in bold indicate which revisions of the folder or item are descendants of the folder or item with the Current (You Are Here)  icon. In other words, the current folder or item is part of the history for the revisions that are in bold.

In the previous two examples, both references were represented in bold text. In the next example, this is not the case. This is because the properties of the *Source Code* folders in both the parent view and the child view have changed. The folder for the parent has revision 1.3, and the folder for the child has revision 1.2.1.0. Both folder histories have gone in different directions.






-  Big Product::Big Product::Big Product\Source Code, 1.3
-  Big Product::BigProduct\Big Product 1.0 Plus Service Packs::Big Product\Source Code, 1.2.1.0

The current folder is a descendant of itself, so it is always represented in bold text. However, it has evolved from the parent folder, so it is no longer in the history of the current folder. Accordingly, the **Folder References** dialog box for the parent folder would present the following information:

-  Big Product::Big Product::Big Product\Source Code, 1.3
-  Big Product::BigProduct\Big Product 1.0 Plus Service Packs::Big Product\Source Code, 1.2.1.0

### File References Created by Branching Views

When you look at the history of a folder or item, you see its ancestors, not its descendants. However, if you change the tip revision in one location and that revision is an ancestor of the tip revision in another location, you might also want to apply your change to the tip revision in the other location (the object of the first descendant). The way to tell if a revision has descendants is to look at its references. Consider the following example showing the references for a file (AUDITSCC.DOC):

-  Help Files::Help Files::Help Files\starteamp::AUDITSCC.DOC, 1.8
-  Help Files::Help Files\Freeze Check::Help Files\starteamp::AUDITSCC.DOC, 1.1
-  Help Files::Help Files\Freeze Check::New View2::starteamp::AUDITSCC.DOC, 1.1.1.0
-  Help Files::Help Files\varc::Help Files\starteamp::AUDITSCC.DOC, 1.6
-  Help Files::Help Files\variant 2::starteamp::AUDITSCC.DOC, 1.2

As the bold text indicates, if the current revision is 1.6, then 1.8 is its only descendant. This also means that you would find revision 1.6 in the history for 1.8.

If a defect is found in revision 1.6 of AUDITSCC.DOC, the bold text helps you determine the descendants of 1.6 that may also need the corrected lines. In this case, 1.8 may need to be updated. The other references are for revisions of the file that:



- Have already diverged (branched) and may be quite different than the current file.
- Are ancestors of the current file and less likely to need a change. For example, they may be in views that are read-only or no longer in use. Whatever the reason for the gap, the ancestors might require far more work than the changes you are about to check in.

You should check for descendants before (and perhaps after) you create a new revision of a folder or item. Before the change becomes a new revision in the application, you can see the descendants. Afterwards, you may see what other references have the same revision number as the newly-changed folder or item. If they, too, have the new revision number, then they, too, already have the new change. For example, the file may be floating in other views.

## References Created by Adding Items to Views

The addition of a new folder or item to a parent or child view can result in one or two references, depending on the relationship between the two views.

If the child view is a branching, floating view, StarTeam creates a reference in each view when a new folder or item is added to the parent.

If the child view is a branching, floating view created using the **Branch None** option, StarTeam creates a reference in each view when a new folder or item is added to the child.

### Floating Down in the View Hierarchy

When a view has a branching child view (whether created with the **Branch None** or **Branch All** option) and the child view is floating, any folder or item added to the parent view becomes visible in both views. The history of the folder or item indicates the view in which the object was created, and the reference hierarchy displays the reference that identifies the parent view as the parent reference.

For example, if a file you add a file to the parent view, its history in either view shows the name of the parent view—until the file branches in the child view.

The following table shows the history in the parent view for a file that was added to the parent view and floated downwards.



**Tip:** You can review historical item information using the **History** tab in the client.


View	Revision	Branch Revision
Big Product	2	1.1
Big Product	1	1.0

The following table shows the history in the child view for a file that was added to the parent view and floated downwards. The history of the file displays the name of the view from which the file was originally added to the application, until the file branches. Then it displays the name of the view in which the file branched.


View	Revision	Branch Revision
branch none floating	3	1.1.1.0
Big Product	2	1.1
Big Product	1	1.0

If you were to display the References tab for this file (`marketshares.doc`) after it has branched in the child view, you would see the following information:

- Big Product::Big Product::Big Product\Marketing Documents::marketshares.doc, 1.1

-  Big Product::`Big Product\branch none floating::Big Product\Marketing Documents::marketshares.doc, 1.1.1.0`

Notice that the history clearly shows the parent view as `Big Product` before the file branches. The history and references for folders and items added to the parent view are similar to those for folders and items that were in the parent view at the time the child view was created.

 **Note:** The name of the views in these examples makes the information easier to understand. You would probably never name a view parent or any other of the names shown these examples.

### Floating Up in the View Hierarchy

When a view has a branching child view (created with the **Branch None** option) and the child view is floating, any folder or item added to the child view becomes visible in both views. This is not true of branching, floating child views that were created using the **Branch All** option.

The history of the folder or item indicates the view in which the object was created, but the reference hierarchy always displays the reference that identifies the parent view as the parent reference.



The following table shows the history in the parent view for a file that was added to a child view and floated upwards. Notice that, even though this is the history in the parent view, the history displays the name of the view from which the file was originally added to the application.

View	Revision	Branch Revision
branch none floating	3	1.2
branch none floating	2	1.1
branch none floating	1	1.0

The following table shows the history in the child view for a file that was added to the child view and floated upwards. The history of the file displays the name of the view from which the file was originally added to the application—until the file branches. Then it displays the name of the view in which the file branched. In this case, those two views just happen to be the same view.

View	Revision	Branch Revision
branch none floating	3	1.1.1.0
branch none floating	2	1.1
branch none floating	1	1.0

When you view the reference hierarchy for a file that floats upwards, you cannot tell that the file was added to the application from the branching child view, and you must investigate the history (using the History tab) of the file to determine where the file originated. For example, the Reference tab would contain the following reference hierarchy for the `slant.doc` file that floated upwards:

-  Big Product::`Big Product::Big Product\Source Code\Timeout::slant.doc, 1.2`
-  Big Product::`Big Product\branch none floating::Big Product\Source Code\Timeout::slant.doc, 1.1.1.0`

### Floating Up and Down in the View Hierarchy

If the view hierarchy is deep (the root view has grandchildren, great-grandchildren, and so on), the use of branching, floating views can cause a great deal of confusion. For example, suppose you add a file to a grandchild of the root view. Further, suppose that this grandchild view was created using the **Branch None** option and that its parent (a child of the root view) was created using the **Branch None** option. The file you add can float up to the parent and grandparent of the current view from which it will, in turn, float back down to the current view. This results in:

- One reference to the file in the current view
- One reference to the file in the parent of the current view (the result of floating up from the current view)
- One reference to the file in the root view (the result of floating up from the parent of the current view)



More references are created if the current view has floating children, grandchildren, and so on. Still more are created if the root view or parent view have other floating children besides the ones mentioned above.

## References Created by Manually Sharing Objects

As you share a folder or item from one location to another (whether in the same view or a different one) an additional reference is created for that object in the new location. The reference for the new folder or item becomes a child of the reference from the folder or item that was shared.

### Reference Hierarchy Example for a Manually Shared File

The following example shows two references for a file named `timeout.cpp`. The file was manually shared from a folder named `Source Code` to a folder named `Timeout` in the same view. Notice that the second reference is based on the first, but created as a by-product of creating a branching view.

-  Big Product::Big Product::Big Product\Source Code::timeout.cpp, 1.0
-  Big Product::Big Product::Big Product\Source Code::Timeout::timeout.cpp, 1.0

The application does not differentiate between references based on what caused them to be created. However, you can tell from the hierarchy that the first reference is the source of the second reference, because the second reference is indented under the first. You can also tell, because they are in the same view, that a manual share or move occurred. (The second reference would be in a different view if it was created automatically when a child view was created.)

A shared folder or item can branch, but may never do so. Regardless, some subset of its history is part of the history of the original folder or item.




### Floating Up and Down in the View Hierarchy

If the view hierarchy is deep (the root view has grandchildren, great-grandchildren, and so on), the use of branching, floating views can cause a great deal of confusion. Suppose all the views except the root view branch and float. At its new location, depending on how views were created, the folder or item you share can float:

- Up the view hierarchy from the recipient view to the root view
- Down to all the recipient children of the view, grandchildren, and so on
- From the recipient view's parents, grandparents, and so on, to all of their other children

This can result in a reference to the folder or item in the new location in every view in the project's view hierarchy. Many of those views may have already had a reference to the folder or item in its old location.

The following example shows all the references created by sharing a file named `shared within child view.doc` from one location in the `branch none floating` view to another location in that same view. The first three references are the references that existed prior to the sharing operation. The fourth reference is the new reference in the root folder. It is shown as a child of the first location in the `branch none floating` view because it floated up from that view. The fifth and sixth references resulted from references that floated down to the `branch none floating` child view of that view.

-  Big Product::Big Product::Big Product\Online Help::shared within child view.doc, 1.0
-  Big Product::Big Product\branch none floating::Big Product\Online Help::shared within child view.doc, 1.0
-  Big Product::Big Product\branch none floating\branch none floating 2::Big Product\Online Help::shared within child view.doc, 1.0

- `Big Product::Big Product::Big Product\Source Code::shared within child view.doc, 1.0`
- `Big Product::Big Product\branch none floating::Big Product\Source Code::shared within child view.doc, 1.0`
- `Big Product::Big Product\branch none floating\branch none floating 2::Big Product\Source Code::shared within child view,1.0`

The next example shows that the file named `shareall.doc` existed only in the *branch all floating* view before it was shared to another view. The reference to the root folder starts the references that occurred as a result of the share operation. However, the recipient view could have been any of the other views, because the file would float up to the root and back down. On the way down, a second reference was created in the *branch all floating* view.

- `Big Product::Big Product::branch all floating::Big Product\Marketing Documents::shareall.doc, 1.0`
- `Big Product::Big Product::Big Product::shareall.doc, 1.0`
- `Big Product::Big Product\branch all floating::Big Product::shareall.doc, 1.0`
- `Big Product::Big Product\branch none floating::Big Product::shareall.doc, 1.0`
- `Big Product::Big Product\branch none floating\branch none floating 2::Big Product::shareall.doc, 1.0`
- `Big Product::Big Product\branch none floating\branch none floating 2\branch none floating 3::Big Product::shareall.doc, 1.0`

## References Created by Moving Objects

When you move a folder or item from one location to another within the same view, StarTeam deletes the object at the old location and reinstates it at the new location. However, there can be side effects in that view's parents and children if any of the views are floating. This is because the copy at the old location is not deleted except in the current view. The parent and child views may end up with two references (one to the old location and one to the new location) instead of one to the new location.

### Reference Hierarchy Examples

Suppose you move the file named `timeout.doc` from the `Marketing Documentation` folder to the `Timeout` folder in a given view that has no branching child views.




The following two examples show the references for this file before and after the move. The number of references is the same; only the path to the file has changed. The file has been deleted from its original location and added to its new location.

Before the move: `Big Product::Big Product::Big Product\Marketing Documents::timeout.doc, 1.0`

After the move: `Big Product::Big Product::Big Product\Source Code\Timeout::timeout.doc, 1.0`

However, suppose this view has a child view that was created without cutting off the connection to the parent (in other words the child view is branched and floating). In the child view, if the moved file has not yet branched, it is not deleted from its old location because you might really still want it here. However, it is added to the new location because it is perceived as a change to the parent that should be reflected in the child.

Notice that the file has only one reference in the parent but that it has two in the child view.

-  Big Product::Big Product::Big Product\Source Code\Timeout::timeout.doc, 1.0
-  Big Product::Big Product\branched floating::Big Product\Marketing Documents::timeout.doc, 1.0
-  Big Product::Big Product\branched floating::Big Product\Source Code \Timeout::timeout.doc, 1.0

Some users sort items using folders. For example, they decide to create a series of folders in a view to classify change requests by criteria such as:

- Will definitely make the next release
- Are under consideration for the next release (time permitting)

These change requests are usually moved from the root folder to one of the sorting folders, or later rearranged and moved from one sorting folder to another. This is a convenience in the current view, but it can cause multiple references in a parent or child view. If the view hierarchy is deep, the current view's parents, grandparents, children, grandchildren, and so on may be affected. Users who use such systems usually create child views that do not float.









### Floating Up and Down in the View Hierarchy


If the view hierarchy is deep (the root view has grandchildren, great-grandchildren and so on), the use of branching, floating views can cause a great deal of confusion. Suppose all the views except the root view branch and float. At its new location, the folder or item you move can float:

- Up the view hierarchy from the recipient view to the root view
- Down to all the children, grandchildren, and so on of the recipient view
- From the parents, grandparents, and so on of the recipient view to all of their other children

A move operation results in one fewer reference to the moved folder or item in the view from which it was moved, and one more reference to it in the recipient view.

The following example shows that a file named *move within parent view* was moved from one location in the root view to another location in that same view (which is why there is only one reference to it in that view). Originally, the file was referenced in five views. The move caused a new reference in all the child views of the root folder, giving each of them two references to the moved file (one reference in its original location and one in its new location).

-  Big Product::Big Product::Big Product\Source Code::moved within parent view.doc, 1.0
-  Big Product::Big Product\branch all floating::Big Product::moved within parent view.doc, 1.0
-  Big Product::Big Product\branch none floating::Big Product::moved within parent view.doc, 1.0
-  Big Product::Big Product\branch none floating\branch none floating2::Big Product::moved within parent view.doc, 1.0
-  Big Product::Big Product\branch none floating\branch none floating2\branch none floating3::Big Product::moved within parent view.doc, 1.0
-  Big Product::Big Product\branch all floating::Big Product\Source Code::moved within parent view.doc, 1.0
-  Big Product::Big Product\branch none floating::Big Product\Source Code::moved within parent view.doc, 1.0
-  Big Product::Big Product\branch none floating\branch none floating2::Big Product\Source Code::moved within parent view.doc, 1.0

-  Big Product::Big Product\branch none floating\branch none floating2\branch none floating3::Big Product\Source Code::moved within parent view.doc, 1.0

## Personal Preferences

The application enables you to set personal preferences that suit your individual work styles. These preferences apply to the currently logged-on user on a given workstation. The **Preferences** command on the **Window** menu allows you to adjust the way the following elements work. If you have other StarTeam clients installed, any analogous option settings made in those clients are automatically set in the StarTeam Eclipse Plugin and vice versa.

## Set of Team Preferences

When you choose to export a set of team preferences in Eclipse, StarTeam provides all the needed IDs, the original server description, and the host and port information. It does not store things settings such as compression or what kind of encryption to use. The user importing the preferences can turn on compression if he or she works remotely or turn on encryption if outside of the firewall. StarTeam automatically sets the encryption level so there is no loss of information.

Also, during an import, you can choose to change the IP addresses on the fly. If a user exported preferences on one machine using internal IP addresses, the user importing may choose to override those with host names and addresses permeable through the firewall.

During an import, you can choose to change the IP addresses of the associated server configurations on the fly. If a user exported preferences on one machine using internal IP addresses, the user importing may choose to override those with host names and addresses permeable through the firewall.

## Setting Preferences

To review a description of the preferences you can set, see the links at the end of this topic.

1. Choose **Window > Preferences** from the main menu. The **Preferences** dialog box opens.
2. In the resulting dialog box, expand **Team > StarTeam**.
3. Select the node containing the options that you want to change.
4. Make the desired changes, then click **OK**.

## StarTeam Preferences


**Window > Preferences > Team > StarTeam**

The general StarTeam preference settings allow you to select a variety of options that affect the way your workstation operates.

Item	Description	Default Setting
Save dirty editors before operations	<p>Before performing an operation with StarTeam, this setting allows you to choose how to handle outstanding edits. Choose one of the following:</p> <ul style="list-style-type: none"> <li>• <b>Never</b> - never save unsaved edits before a StarTeam operation begins.</li> <li>• <b>Prompt</b> - the application prompts you whether to save unsaved edits</li> </ul>	Prompt

Item	Description	Default Setting
Synchronize, check-in, or check-out after sharing	<p>before a StarTeam operation begins.</p> <ul style="list-style-type: none"> <li>• <b>Auto-save</b> - automatically saves edits before a StarTeam operation begins.</li> </ul> <p>Specifies the default synchronization, check-in, or check-out action after sharing. Choose one of the following:</p> <ul style="list-style-type: none"> <li>• <b>Never</b> - never synchronize after a StarTeam operation begins.</li> <li>• <b>Prompt</b> - the application prompts you whether to synchronize after a check-in or check-out operation.</li> </ul>	Prompt
Comment History	Specifies the maximum number of comments to retain.	30
Connect to StarTeam Server	<p>Specifies the default action to take when selecting a server in the Server Explorer. Choose from:</p> <ul style="list-style-type: none"> <li>• <b>Never</b> - never connect to the server. Work offline.</li> <li>• <b>Prompt</b> - always prompt the user whether to connect to the server.</li> <li>• <b>Always</b> - always connect to the server without prompting the user.</li> </ul>	Prompt
Property editors style	<p>Specifies which style of property editor opens when double-clicking a selected item in the item view. Choose from:</p> <ul style="list-style-type: none"> <li>• <b>Embedded editor</b> - double click opens the selected item in the non-modal embedded editor.</li> <li>• <b>Property dialog</b> - double click opens the selected item in the modal property dialog box.</li> <li>• <b>Wrap text in multi-line text fields</b></li> <li>• <b>Use new date/time controls</b> - displays the Eclipse calendar controls instead of combo boxes.</li> <li>• <b>Require confirmation on dirty editors cancellation</b></li> <li>• <b>Embedded alternate property editors</b> - Allow alternate property editors to open as embedded editors.</li> </ul>	Not specified
Link view with selection	When opening a connection to a server configuration, all StarTeam views are updated to reflect the contents of the currently selected server configuration in the Server Explorer. Depending on how much	All views selected

Item	Description	Default Setting
	information is in the repository, it can take some time to load all of this information. This setting allows you to check or clear the views that you want updated.	

 **Note:** Click **Restore Defaults** to restore the default values for these settings.

## Change Request Options

**Window > Preferences > Team > StarTeam > Change Request**

Use the **Change Request** options to specify the criteria that the application uses to determine whether a change request has been read. You can also indicate how often the application should search for new change requests and how change request locking issues should be handled.

### Mark as read

#### When change request is selected

Marks an asset as read as soon as you select it. An unread asset is displayed with bold font. A read asset is displayed with regular font.

#### When selected for \_\_\_ seconds

Marks an asset read after it has been selected for the specified number of seconds. The range is from 15 to 9999 seconds.

#### Only when manually marked as read

Marks a selected asset read when you choose **Change Request > Mark as Read**.



**Note:** Assets are always marked as read when you display their properties.

### System tray notifications

#### Check for new or modified change requests

Checks for new or modified assets at regular intervals and lets you know that you have become responsible for new assets. If this option is checked, you must also specify the number of minutes between checks in the Interval option below. When unchecked, the application does not place an icon in the system tray for a new change request.

#### Interval (in minutes)

Specifies the number of minutes between automatic checks for new or modified assets. The default is 10 minutes.

### Locking

#### Exclusively lock change request during edit

Locks an asset when you open its **Properties** dialog box for editing. If unchecked, the application does not lock assets when you open its **Properties** dialog box.

#### Clear manually locked change requests after edit

Unlocks a locked asset after you have edited its properties and clicked **OK** to create a new revision. If unchecked, the application does not remove the locks.

### URL Options

#### Display template

Specifies a special template used to generate an HTML representation of an item when the item's URL is copied to the Clipboard. With no format, there is a default HTML representation that specifies the type of item and identifies it by name and number. When the text is generated from the template, the specified property values are substituted for the variables in `~~*~~`. The variables may be referenced by the same names used in report templates, as well as by the display name of the property. When using the display name, you can omit spaces, and case



will be ignored. For example, if you use the following sample template for a file: `Change Request:~~CR Number~~:~~CreatedBy~~`, the HTML representation will be `Change Request:38,849:Tom Smith`. This template is a superset of that used by the Report feature of the client.

**Generate ID-based URLs** Specifies the URL by ID rather than by name. For example, an ID-based URL would be `starteam://hostname:49201/12;ns=Project;scheme=id`, while a name-based URL would be `starteam://hostname:49201/myproject`.



**Note:** Folders always use an ID-based URL.




**Note:** If you do not select either of the locking options, opening a change request will not lock it; you must manually lock and unlock it. If you select the **Exclusively Lock** option only, change requests that are not already locked become locked when you open them and unlocked when you click **Cancel** or **OK**. If you select the **Clear change request Locks** option only, any change request that you have locked manually becomes unlocked when you click **OK** to create a new revision. If you select both options, you can lock change requests manually or by opening them. These change requests become unlocked when you click **OK** to create new revisions or (if they were not locked prior to being opened) when you click **Cancel**.

## Console Options

**Window > Preferences > Team > StarTeam > Console**

The Console preference settings allow you to configure the StarTeam console. You can set properties for output colors and text font and configure the console to open automatically when the application generates output.

<b>Enable console</b>	When checked, displays the console window.
<b>Fixed width console</b>	When enabled, specifies the width of lines output to the console. Enabling this option allows you to specify the console width using the <i>Character width</i> preference setting.
<b>Character width</b>	Specifies the width of lines output to the console. You can specify this option when activating the <i>Fixed width console</i> preference.
<b>Limit console output</b>	Limits the number of characters buffered by the console. Use this option with the <i>Console buffer size (in characters)</i> preference to set the console buffer size.
<b>Console buffer size (in characters)</b>	Use this option with the <i>Limit console output</i> preference to set the number of characters buffered by the console.
<b>Show StarTeam console automatically when command is run</b>	Shows the output of StarTeam commands in the Console view.  <b>Note:</b> With this option enabled, you may experience slower command operation.
<b>Console text color settings</b>	You can customize the text colors in the StarTeam console for the following: <ul style="list-style-type: none"><li>• Command line</li><li>• Message</li><li>• Error</li></ul>

## File Options

Window > Preferences > Team > StarTeam > File

Use **File Options** to customize the way you work with files. In a few cases (such as Marking Unlocked Files Read Only), your administrator's choices may override your preferences.

<b>Check-out</b>	<b>Use last modification time for check-out files</b>	Uses the same time for each checked-out file as the time stamp of the revision being checked out. Otherwise, the time stamp used for the checked-out file is the current time (the time check-out occurs.)
<b>Merging</b>	<b>Always pop-up merge utility</b>	Opens the merge utility to display the merged file even when there are no conflicts.
	<b>Pop-up merge utility in case of conflicts only</b>	Opens the merge utility only if the merged file contains conflicts. If unchecked, files will be checked in and out automatically.
<b>Deleted Status Values</b>	<b>Show 'Deleted' File Status Values</b>	Shows the status values for deleted files.
<b>Locking</b>	<b>Exclusively lock files on check-out</b>	Sets the default <b>Lock Status</b> option to <b>Exclusive</b> in the <b>Check Out</b> dialog box . Otherwise, the default is <b>Keep Current</b> .
	<b>Clear file locks on check-in</b>	Sets the default <b>Lock Status</b> option to <b>Unlock</b> in the <b>Check Out</b> dialog box. Otherwise, the default is <b>Keep Current</b> .
	<b>Use non-exclusive locks in integration</b>	Creates a non-exclusive lock when locking a file from the application integration– that is, a lock that allows others to check in the file. Using non-exclusive locks also allows more than one person to edit a file at one time. If team members are not editing the same lines of the file, the merged file usually has no conflicts.
	<b>Mark unlocked working files read-only</b>	Sets working copies of files that you have not locked to read-only when you add files, check in files, check out files, or unlock files. If this check box is selected, only locked files can be edited.
<b>EOL</b>	<b>Automatic EOL conversion for check-out operations</b>	<p>Performs an automatic EOL conversion on check-out operations. If checked, also select the operating system on which you are working:</p> <ul style="list-style-type: none"><li>• Windows ( CR-LF )</li><li>• Unix ( LF )</li><li>• Mac ( R )</li></ul> <p>Files can be checked out in <b>LF</b> format on every platform, regardless of specific options.</p> <p>The EOL Format property can be set in the StarTeam Cross-Platform Client in the <b>Add/Check-in</b> and <b>File Properties</b> dialog boxes.</p> <p>The default for automatic EOL conversion for check-out operations is “checked” if the user does not have that option defined already.</p> <p>The EOL Property values are:</p>

<b>Undefined</b>	(null in the SDK).
<b>Client Defined</b>	Causes workstation default or per-checkout EOL conversion option to be used.
<b>Fixed CR, Fixed LF, and Fixed CRLF</b>	Causes this EOL format to be used always. The workstation/check-out conversion option is ignored.




**Note:** Once EOL Format is defined, **Update Status** works for all text files, regardless of what EOL format was used when they were checked-out. For compatibility with older Clients, if check-out "EOL conversion" is not requested, and EOL Format is Undefined, files are still checked out with the EOL convention with which they were added to the StarTeam Server.

## General

<b>Use file checksums (MD5) to calculate status</b>	Uses the checksum instead of the file time stamp and size to compute the <b>Status</b> field when the application is refreshed. Using the checksum provides a more accurate status value than the time stamp, but takes longer. If unchecked, the application uses the time stamp and size.
<b>File encoding for keyword expansion</b>	Specifies the code page to be used for keyword expansion by choosing a default file encoding from the list.

## Repository

<b>File status repository default</b>	Indicates where you want file status information stored, either in a central repository location on your workstation or in a child folder (named <code>.sbas</code> ) of each working folder.
<b>Central</b>	You can enter or browse for a location on your computer other than the default central repository location. Whenever you make a change to a file in the working folder, the status for that file is undated only on your computer in the specified location. Everyone else sees the status <b>Unknown</b> for that file. Over time, all the files may have been changed, and the statuses can become <b>Unknown</b> for all users of all files.
<b>Per-folder</b>	Useful in the special case where multiple users are sharing a working folder, for example, on a shared network drive. For example, suppose several users all check files in and out of a shared working folder. If these users have set the central repository option for file statuses, the statuses are stored on each of their computers. Whenever a user makes a change to a file in the working folder, the status for that file is undated only on that user's computer. Everyone else sees the status <b>Unknown</b> for that file. Over time, all the files may have been changed, and the statuses can become <b>Unknown</b> for all users of all files. Using the per-folder option causes the statuses to be updated within the working folder itself. Everyone has access to those status changes and <b>Unknown</b> statuses do not occur.

	<b>Purge</b>	Opens the <b>Status Repository Cleanup</b> dialog box where you can remove file status data from the workstation status repository.
	<b>Default</b>	Resets the <b>Central</b> repository location to the default setting
<b>URL Options</b>	<b>Display template</b>	Specifies a special template used to generate an HTML representation of an item when the item's URL is copied to the Clipboard. With no format, there is a default HTML representation that specifies the type of item and identifies it by name and number. When the text is generated from the template, the specified property values are substituted for the variables in <code>~~*~~</code> . The variables may be referenced by the same names used in report templates, as well as by the display name of the property. When using the display name, you can omit spaces, and case will be ignored. For example, if you use the following sample template for a file: <code>~~FolderPath~~:~~Name~~</code> , the HTML representation will be the path to the selected file: <code>StarTeam\:\buildinfo.properties</code> . This template is a super-set of that used by the report feature of the client.
	<b>Generate ID-based URLs</b>	Specifies the URL by ID rather than by name. For example, an ID-based URL would be <code>starteam://hostname:49201/12;ns=Project;scheme=id</code> , while a name-based URL would be <code>starteam://hostname:49201/myproject</code> .
		<b>Note:</b> Folders always use an ID-based URL.
<b>Alternate Applications</b>		Opens the <b>Alternate Applications</b> dialog box where you can specify an alternate editor, merge utility, and comparison utility to use in the application if you don't want to use the default tools for those functions. Includes fields for specifying options to use with the applications.
	<b>Open With...</b>	Enables you to provide a command on a non-Microsoft Windows system that will display at least one type of files and folders. The command should consist of the path to an application and the command-line options for which the application for which the application can substitute the selected file. The application runs this command whenever you do one of the following: Double-click a file or folder in the item list, double-click an attachment, or generate and open a report.  The following command is suggested: <code>netscape -remote "openFile(\$file)"</code> because Netscape can handle many different media types, such as image files, text files, and HTML.
	<b>Merge Utility Options</b>	Use the following command-line options to represent files sent to the alternate merge utility. <ul style="list-style-type: none"> <li><b>\$branchtip</b> A place holder for the path to the tip revision of the file to be merged.</li> <li><b>\$usertip</b> A place holder for the path to the local working file to be merged.</li> <li><b>\$basefile</b> A place holder for the path to the common ancestor for the <code>\$branchtip</code> and <code>\$usertip</code> files.</li> <li><b>\$resultfile</b> A place holder for the path to the file that will store the output from the merged file.</li> </ul>

<b>Compare Utility Options</b>	Use the following command-line options to represent files sent to the alternate compare utility.
<b>\$file1</b>	A place holder for the path to the first of the two files to be compared.
<b>\$file2</b>	A place holder for the path to the second of the two files to be compared.

## Folder Options

**Window > Preferences > Team > StarTeam > Folder**

The folder options enable you to specify the criteria that the application uses to determine whether a folder has been read. You can also indicate how often the application should search for new folders and how folder locking issues should be handled.

<b>Check-out</b>	<b>Use Last Modification Time for Folders During Checkout</b>	Uses the last modification time when Working folders are created. Otherwise it uses the current time.
<b>Locking</b>	<b>Exclusively Lock Folder During Edit</b>	Locks an asset when you open its <b>Properties</b> dialog box for editing. If unchecked, the application does not lock assets when you open its <b>Properties</b> dialog box.
	<b>Clear Manually Locked Folders After Edit</b>	Unlocks a locked asset after you have edited its properties and clicked <b>OK</b> to create a new revision. If unchecked, the application does not remove the locks.



**Note:** If you do not select either of the locking options, opening a change request will not lock it; you must manually lock and unlock it. If you select the **Exclusively Lock** option only, change requests that are not already locked become locked when you open them and unlocked when you click **Cancel** or **OK**. If you select the **Clear change request Locks** option only, any change request that you have locked manually becomes unlocked when you click **OK** to create a new revision. If you select both options, you can lock change requests manually or by opening them. These change requests become unlocked when you click **OK** to create new revisions or (if they were not locked prior to being opened) when you click **Cancel**.

<b>URL Options</b>	<b>Display template</b>	Specifies a special template used to generate an HTML representation of an item when the item's URL is copied to the Clipboard. With no format, there is a default HTML representation that specifies the type of item and identifies it by name and number. When the text is generated from the template, the specified property values are substituted for the variables in <code>~~*~~</code> . The variables may be referenced by the same names used in report templates, as well as by the display name of the property. When using the display name, you can omit spaces, and case will be ignored. For example, if you use the following sample template for a file: <code>~~FolderPath~~:~~Name~~</code> , the HTML representation will be the StarTeam path to the selected folder: <code>SampleProject\ReadMe</code> . This template is a super-set of that used by the Report feature of the client.
	<b>Generate ID-based URLs</b>	Specifies the URL by ID rather than by name. For example, an ID-based URL would be <code>starteam://hostname:49201/12;ns=Project;scheme=id</code> , while a name-based URL would be <code>starteam://hostname:49201/myproject</code> .



**Note:** Folders always use an ID-based URL.

**Show Not-in-View Folders By Default** Checks the **Show Not-in-View Folders** on the **Folder Tree** menu to set it to be on by default. (Changing this check box does not affect projects that are already open.)

## Mapping Preferences

**Window > Preferences > Team > StarTeam > Folder > Mapping**



**Note:** You can set folder mappings in the client using the **Mapping** options in the **Preferences** dialog box or at the project-level using the **Properties** menu found in the Eclipse Explorers on shared projects.

The Mapping preference settings allow you to map repository folders to your Eclipse workspace folders. These preferences enable you to change the naming of local folders taken from either the human-readable names of StarTeam folders or their path fragments. These options correspond to using *default* or *alternate* working folders in the Cross-Platform Client.



**Note:** Most users should use the default setting. It is not advisable to change the folder mappings on-the-fly. You should keep your folder mappings consistent for the duration of the workspace.

You can customize the following mapping options at the default and project-specific levels:

- Map name (and replace non-path characters by '-'): The default setting.
- Map path fragment: The setting used to indicate alternate folder names.

You do not have to be concerned with Mapping preferences for project, view, or root folder names. These names are associated with internal identifiers (IDs), so there is no need to use any type of Mapping preferences. You can name your local project whatever you like. Mapping does not apply at all in this case, and the default name suggested for the local project is the root folder name of the view; however, since the project, view, and folder IDs are used for sharing, the local project name can be changed at your own discretion.

However, the client handles local folder names differently. While the relationship between local folders and remote (server-side) folders is established by an ID, on an initial folder mapping or a local folder creation and naming, the Mapping Preferences apply for how the client identifies folders that have not yet been shared and finds suitable remote folders for them.

Path fragments are server-side only and do not change any property for the client. However, if you choose in the client to use path fragments for the workspace folder naming, you may need to refresh the latter to be aware of the remote changes. A local, automatic adjustment does not exist if you change the path fragment so drastically that the local folder is no longer named accordingly. If this happens, the local folder will lose its relationship to its remote peer despite the folder ID it remembers and show up as a local addition in the client.



**Note:**

- By default, StarTeam uses remote (server-side) folder names for the local workspace folder names. The standard mapping setting is not to allow mapping at all. However, you can change the Mapping options to use the folder path fragment. Relative paths (`../x/y/z`) are not accepted. Alternatively, the mapping can default to the last segment of the path fragment.
- You can check-in the Eclipse `<project>/.settings/com.microfocus.team.starteam.provider.prefs` file to share the same folder mapping settings for that project across teams. To provide general mapping coherence, you can export and share the preferences on a file server for importing by team members.

**Map name (and replace non-path characters by '-')** The default setting. Maps to the *default* setting for StarTeam Server-side (remote) folders. Most users will use this as their default setting. This means that you are using the default folder name provided by the Server to name your local folders.

**Map working folder** Use this option to set *alternate* working folders. You can choose from the following options to handle an absolute or relative working folder path:

- **Default to last segment** - Use this setting to keep the last segment of the path. This is helpful for users that redirect folders to other locations without actually changing the last segment of the working folder path from the initial short working folder. For example, you may want to use this option for folders that you never perform a check-in on, such as a folder that you locally name as, *plugins[DO NOT CHECK IN]*.
- **Report error** - Does not allow remote folders with complex working folder paths to be mapped. This option keeps you from checking out folders that are pointing to other locations other than the default setting on the Server.

**Configure project specific settings** You can overload preferences on a per-project basis. This link opens the **Project Specific Configuration** dialog box where you can select a project and configure mapping preferences specifically for it.

Overloaded preference settings are stored within a project in the path ".settings/com.microfocus.team.starteam.provider.prefs". You can check them in and share them across teams to enforce the same folder mapping for every team member.

### Folder Mapping Examples

Imagine a StarTeam project that has the following folder structure (folder name in quotes, working folders in parenthesis):

```
"<My Project>" (C:\prj)
  "CRs/Tasks/Requirements" (process)
  "Documentation" (../html/docs)
  "Source" (S:\src\java)
```

The three mapping methods create the following workspace structures:

```
Map name (and replace non-path characters by '-') ... "comments
in quotes"
```

```
    _My Project_ ... "the '<' and '>' characters in the
root folder name get replaced by '_' to comply with eclipse
resource naming rules"
```

```
    CRs_Tasks_Requirements ... "the '/' characters in
this folder name get replaced by '_' to comply with platform
file and folder naming rules"
```

```
    Documentation ... "no conversion needed for this
folder name"
```

```
    Source ... "no conversion needed for this folder
name"
```

```
Map working folder ...
```

```
... If absolute or relative working folder is set ...
```

```
... Default to last segment
```

```
    prj ... "the last segment of the absolute path in the
```

```

view '<My Project>' working folder 'C:\prj' is used"
    docs ... "the last segment of the relative path in
the 'Documentation' folder's working folder '../html/docs' is
used"
    java ... "the last segment of the absolute path on
another drive in the 'Source' folder's working folder 'S:\src
\java' is used"
    process ... "the working folder of folder 'CRs/
Tasks/Requirements' can be taken unchanged"

... Report error ()

    An exception would be thrown at sharing, indicating
the first absolute path used for the view's working folder as
non-mappable.

```

The Cross-Platform Client allows folders with an empty working folder property. Such folders will not map into Eclipse workspaces when using working folder mapping, and a thrown exception prevents projects with such folder structures from synchronizing properly.



**Note:** The workspace project name does not need to be the same as the folder name and working folder path. The sharing parameters are ID-based to allow both remote and local renaming without losing the sharing. Nonetheless, wizards that check out repository folders as workspace projects observe the mapping preferences, and choose the suggested project name based on the general mapping preferences. If mapping is not configured to use folder names, the last segment of the working folder is suggested for project naming.

## Label Decorations Options

### Window > Preferences > Team > StarTeam > Label Decorations

Label Decorations show extra information about an item on its label or icon. The preferences enable you customize the text, font, and icons associated with your projects, folders, and files. You can indicate what text label flags you want to use for various file states and which variables should display with files, folders, and projects.



**Note:** To display label decorations, the **StarTeam Provider** and **StarTeam Server Explorer** options must be checked in the **General > Appearance > Label Decorations** options of the **Preferences** dialog box.

#### General

Use the options on this page to configure general preferences about the decorators:

**Enable color decoration** - This option enables color decorations. You can configure the colors used for incoming changes, outgoing changes, invisible folders, and other changes using the **General > Appearance > Colors and Fonts** preference page.

**Enable font decoration** - This option enables font decorations. You can configure the fonts used for incoming changes, outgoing changes, invisible folders, and other changes using the **Colors and Fonts** preference page. When you activate *Enable font decoration*, the application also activates the following options:

- Decorate incoming changes
- Decorate outgoing changes
- Decorate bidirectional changes

#### Text Decorations

These settings specify how StarTeam information displays on text labels. Use the *File Decoration*, *Folder Decoration*, and *Project Decoration* options to edit the variable(s) that display for a file, folder, or project and the order in which they should display. To add a



variable to the format, click **Add Variables**, select the desired variable(s) from the list, and click **OK**. Use the *Branch on check-in flag* to enter your preferred text indicator.

Default setting:

```
File Decoration: ${name} (${dot_notation})
Folder Decoration: ${name}
Project Decoration: ${name} [${view_path}]
Branch on check-in flag: !
```

## Icon Decorations

These settings specify the overlay icons used to show StarTeam-specific information in views. You can activate icon decorations for the following:

- Current
- Outgoing changes
- Incoming changes
- Bidirectional changes
- Locked exclusively
- Excluded

The **Preview** pane at the bottom of the page shows you the various icon and text decorations associated with the various states of StarTeam items.

## Label Decorations (for Provider) Preferences

**Window > Preferences > Team > StarTeam > Label Decorations (for Provider)**

Label Decorations show extra information about an item on its label or icon. In your StarTeam Label Decorations (for Provider) preferences, you can indicate what text label flags you want to use for various file states and which variables should display with files, folders, and projects.



**Note:** To display label decorations, the **StarTeam Provider** and **StarTeam Server Explorer** options must be checked in the **General > Appearance > Label Decorations** options of the **Preferences** dialog box.

### General

Use the options on this page to configure general preferences about the decorators:

**Enable color decoration** - This option enables color decorations. You can configure the colors used for incoming changes, outgoing changes, invisible folders, and other changes using the **General > Appearance > Colors and Fonts** preference page.

**Enable font decoration** - This option enables font decorations. You can configure the fonts used for incoming changes, outgoing changes, invisible folders, and other changes using the **General > Appearance > Colors and Fonts** preference page. When you activate *Enable font decoration*, the application also activates the following options:

- Decorate incoming changes
- Decorate outgoing changes
- Decorate bidirectional changes

### Text Decorations


These settings specify how StarTeam information displays on text labels. Use the *File Decoration*, *Folder Decoration*, and *Project Decoration* options to edit the variable(s) that display for a file, folder, or project and the order in which they should display. To add a variable to the format, click **Add Variables**, select the desired variable(s) from the list, and click **OK**. Use the *Branch on check-in flag* to enter your preferred text indicator.

Default setting:

```
File Decoration: ${name} (${dot_notation})
Folder Decoration: ${name}
Project Decoration: ${name} [${view_path}]
Branch on check-in flag: !
```

**Icon Decorations** These settings specify the overlay icons used to show StarTeam-specific information in views. You can activate icon decorations for the following:

- Current
- Outgoing changes
- Incoming changes
- Bidirectional changes
- Locked exclusively
- Excluded

 **Tip:** The **Preview** pane at the bottom of the page shows you the various icon and text decorations associated with the various states of StarTeam items.

## Label Decorations (for Server Explorer) Preferences

**Window > Preferences > Team > StarTeam > Label Decorations (for Server Explorer)**

Label Decorations show extra information about an item on its label or icon. In your StarTeam Label Decorations preferences, you can indicate what text label flags you want to use for various file states and which variables should display with files, folders, and projects.



**Note:** To display label decorations, the **StarTeam Provider** and **StarTeam Server Explorer** options must be checked in the **General > Appearance > Label Decorations** options of the **Preferences** dialog box.

**General** Currently, there are no general preferences to set for the Server Explorer.

**Text Decorations** These settings specify how StarTeam information displays on text labels. Use decoration field to edit the variable(s) that display for the corresponding item, view, project, or server, and to specify the order in which they should display. To add a variable to the format, click **Add Variables**, select the desired variable(s) from the list, and click **OK**. Use the *Branch on check-in flag* to enter your preferred text indicator.

Default setting:

```
Task Decoration: ${name}
Topic Decoration: ${name}
Requirement Decoration: ${name}
File Decoration: ${name} (${property:DotNotation}) Locker: $
{property:ExclusiveLocker}
Folder Decoration: ${name}
Project Decoration: ${name}
Server Decoration: ${name} [${property:LoggedInUser}]
```

**Icon Decorations** These settings specify the overlay icons used to show StarTeam-specific information in views. You can activate icon decorations for the following:

- Not-in-view Folder
- Locked exclusively
- Flagged
- Task status



**Tip:** The **Preview** pane at the bottom of the page shows you the various icon and text decorations associated with the various states of StarTeam items.

## Password Management Preferences

**Window > Preferences > Team > StarTeam > Password Management**

The Password Management preference settings allow you to view the stored information for your configured StarTeam repository locations and enables you to delete this information.



**Note:** StarTeam uses a standard Eclipse feature to store password information. The Eclipse encryption tool keeps password information securely hidden.

**Connection** This field displays the connection name that you supply in the Server description field when creating a connection to a StarTeam server configuration.

**User** This field displays the user currently logged in to the specified server configuration.

**Status** This field indicates whether the user is logged on or off of the specified server configuration.

**Remove** Select one row from the list, and click **Remove** to delete the stored information.

**Remove All** Deletes all stored information.

## Requirements Options

**Window > Preferences > Team > StarTeam > Requirement**

Use the **Requirement Options** to specify the criteria that the application uses to determine whether a requirement has been read. You can also indicate how often the application should search for new requirements and how requirement locking issues should be handled.

### Mark as read

**When requirement is selected**

Marks an asset as read as soon as you select it. An unread asset is displayed with bold font. A read asset is displayed with regular font.

**When selected for \_\_\_ seconds**

Marks an asset read after it has been selected for the specified number of seconds. The range is from 15 to 9999 seconds.

**Only when manually marked as read**

Marks a selected requirement read when you choose **Requirement > Mark as Read**.



**Note:** Assets are always marked as read when you display their properties.

### System tray notifications

**Check for new or modified requirements**

Checks for new or modified assets at regular intervals and lets you know that you have become responsible for new assets. If this option is checked, you must also specify the number of minutes between checks in the Interval option below. When unchecked, the application does not place an icon in the system tray for a new change request.

**Interval (in minutes)**

Specifies the number of minutes between automatic checks for new or modified assets. The default is 10 minutes.

### Locking

**Exclusively lock requirement during edit**

Locks an asset when you open its **Properties** dialog box for editing. If unchecked, the application does not lock assets when you open its **Properties** dialog box.

**Clear manually locked requirements after edit**

Unlocks a locked asset after you have edited its properties and clicked **OK** to create a new revision. If unchecked, the application does not remove the locks.



**Note:** If you do not select either of the locking options, opening a change request will not lock it; you must manually lock and unlock it. If you select the **Exclusively Lock** option only, change requests that are not already locked become locked when you open them and unlocked when you click **Cancel** or **OK**. If you select the **Clear change request Locks** option only, any change request that you have locked manually becomes unlocked when you click **OK** to create a new revision. If you select both options, you can lock change requests manually or by opening them. These change requests become unlocked when you click **OK** to create new revisions or (if they were not locked prior to being opened) when you click **Cancel**.

**URL Options**

**Display template**

Specifies a special template used to generate an HTML representation of an item when the item's URL is copied to the Clipboard. With no format, there is a default HTML representation that specifies the type of item and identifies it by name and number. When the text is generated from the template, the specified property values are substituted for the variables in `~~*~~`. The variables may be referenced by the same names used in report templates, as well as by the display name of the property. When using the display name, you can omit spaces, and case will be ignored. For example, if you use the following sample template for a file:  
`Requirement #~~Number~~: ~~Status~~`, the HTML representation will be `Requirement #34,132: Submitted`. This template is a superset of that used by the Report feature of the client.

**Generate ID-based URLs**

Specifies the URL by ID rather than by name. For example, an ID-based URL would be `starteam://hostname:49201/12;ns=Project;scheme=id`, while a name-based URL would be `starteam://hostname:49201/myproject`.



**Note:** Folders always use an ID-based URL.

## Sprint Options

**Window > Preferences > Team > StarTeam > Sprint**

Use the **Sprint** options to specify when the application looks for new or modified sprints.

**System tray notification**

**Check for new or modified sprints**

Checks for new or modified assets at regular intervals and lets you know that you have become responsible for new assets. If this option is checked, you must also specify the number of minutes between checks in the Interval option below. When unchecked, the application does not place an icon in the system tray for a new change request.

**Interval (in minutes)**

Specifies the number of minutes between automatic checks for new or modified assets. The default is 10 minutes.

<b>URL Options</b>	<b>Display template</b>	Specifies a special template used to generate an HTML representation of an item when the item's URL is copied to the Clipboard. With no format, there is a default HTML representation that specifies the type of item and identifies it by name and number. When the text is generated from the template, the specified property values are substituted for the variables in <code>~~*~~</code> . The variables may be referenced by the same names used in report templates, as well as by the display name of the property. When using the display name, you can omit spaces, and case will be ignored. For example, if you use the following sample template for a sprint: <code>sprint # ~sprintnumber~: ~Title~, Status - ~status~</code> , the HTML representation will be <code>Sprint #34,132: Sprint Title, Status - Active</code> . This template is a super-set of that used by the report feature of the client.
<b>Generate ID-based URLs</b>		Specifies the URL by ID rather than by name. For example, an ID-based URL would be <code>starteam://hostname:49201/12;ns=Project;scheme=id</code> , while a name-based URL would be <code>starteam://hostname:49201/myproject</code> .



**Note:** Folders always use an ID-based URL.

## StarTeamMPX Options

**Window > Preferences > Team > StarTeam > StarTeamMPX**

Servers that use offer additional caching services and performance enhancements. To take advantage of these benefits, must be enabled and configured on your workstation so that any open project view can take advantage of .

The right end of the application status bar displays the current status of on your workstation. The words and icons on the status bar for are as follows:

<b>Yellow lightning bolt</b>	Indicates that MPX is available and enabled for the currently selected project view.  If Web Edition can use the default client profile for an MPX-enabled server and, therefore, take advantage of MPX, this icon appears in front of the server configuration's name in the browser window.
<b>Gray lightning bolt</b>	Indicates that MPX is available for the currently selected project view but that it has not been enabled in the client.
<b>Red circle with a slash beside a yellow lightning bolt</b>	Indicates that MPX is enabled for the currently selected project view, but something happened to break the connection. For example, the Message Broker may be stopped.
<b>(no icon)</b>	Indicates that MPX is not available for the currently selected project view.
<b>Instant</b>	Indicates that MPX's auto-refresh is turned on.
<b>Auto</b>	Indicates that your workstation's auto-refresh is turned on, but that MPX's auto-refresh is either turned off or unavailable. (Your workstation's auto-refresh option is on the <b>Workspace</b> tab of the <b>Personal Options</b> dialog box.)
<b>Manual</b>	Indicates that your workstation's auto-refresh is turned off and that MPX's auto-refresh is either turned off or unavailable. You must manually refresh the current project view by pressing F5.

## Options

**Enable** Enables your workstation to use StarTeamMPX if it is available on the server. Changing this check box does not affect open projects. StarTeamMPX is enabled by default.

**Automatic refresh with** Enables automatic refresh of the application window by way of MPX, with options for setting the minimum and maximum delay times between refreshes. The default minimum is 30 seconds, and the default maximum is 0– seconds. If this option is unchecked, you must refresh manually ( **Shift +F5** .)

When **Automatic Refresh** is enabled, after every change to the view, StarTeam waits a minimum number of seconds before refreshing. That means that if changes are infrequent, the application performs a refresh almost immediately. However, if changes are frequent, the minimum refresh timer is constantly being reset and never reaches the number of seconds set for a refresh. In such cases, the next refresh occurs when the maximum number of seconds between refreshes forces a refresh.

Automatic refresh is designed to perform even if the client is minimized. When **Automatic refresh** enabled, if the network is down, **Automatic refresh** will attempt to refresh the data in the client resulting in a connection error.

<b>Use MPX Cache Agent for</b>	<b>File Content</b>	Use the MPX Cache Agent to retrieve file content.
	<b>Item Properties</b>	Use the MPX Cache Agent to retrieve item properties.
	<b>Use MPX Cache Agent at</b>	Designates a specific MPX Cache Agent to use by IP address and port number.
	<b>Automatically locate the closest MPX Cache Agent for Server</b>	Locates the network nearest MPX Cache Agent automatically, but only if the server is MPX-enabled.
	<b>Maximum request threads</b>	Specifies the maximum number of request threads allowed. The default is 2, and 2 to 3 should be adequate for most of your needs.

## Story Options

**Window > Preferences > Team > StarTeam > Story**

Use the **Story** options to specify when the application looks for new or modified stories.

<b>System tray notification</b>	<b>Check for new or modified stores</b>	Checks for new or modified assets at regular intervals and lets you know that you have become responsible for new assets. If this option is checked, you must also specify the number of minutes between checks in the Interval option below. When unchecked, the application does not place an icon in the system tray for a new change request.
	<b>Interval (in minutes)</b>	Specifies the number of minutes between automatic checks for new or modified assets. The default is 10 minutes.
<b>URL Options</b>	<b>Display template</b>	Specifies a special template used to generate an HTML representation of an item when the item's URL is copied to the Clipboard. With no format, there is a default HTML representation that specifies the type of item and identifies it by name and number. When the text is generated from the

template, the specified property values are substituted for the variables in `~~*~~`. The variables may be referenced by the same names used in report templates, as well as by the display name of the property. When using the display name, you can omit spaces, and case will be ignored. For example, if you use the following sample template for a story: `story # ~storynumber~: ~Title~, Status - ~status~`, the HTML representation will be `Story #34,132: Story Title, Status - Active`. This template is a superset of that used by the report feature of the client.

**Generate ID\_based URLs** Specifies the URL by ID rather than by name. For example, an ID-based URL would be `starteam://hostname:49201/12;ns=Project;scheme=id`, while a name-based URL would be `starteam://hostname:49201/myproject`.



**Note:** Folders always use an ID-based URL.

## Task Options

Window > Preferences > Team > StarTeam > Task

Use the **Task Options** to specify the criteria that the application uses to determine whether a task has been read. You can also indicate how often the application should search for new tasks and how task locking issues should be handled.

### Mark as read

#### When task is selected

Marks an asset as read as soon as you select it. An unread asset is displayed with bold font. A read asset is displayed with regular font.

#### When selected for \_\_\_ seconds

Marks an asset read after it has been selected for the specified number of seconds. The range is from 15 to 9999 seconds.

#### Only when manually marked as read

Marks a selected task read when you choose **task > Mark as Read**.



**Note:** Assets are always marked as read when you display their properties.

### System tray notifications

#### Check for new or modified tasks

Checks for new or modified assets at regular intervals and lets you know that you have become responsible for new assets. If this option is checked, you must also specify the number of minutes between checks in the Interval option below. When unchecked, the application does not place an icon in the system tray for a new change request.

#### Interval (in minutes)

Specifies the number of minutes between automatic checks for new or modified assets. The default is 10 minutes.


### Locking

#### Exclusively lock task during edit

Locks an asset when you open its **Properties** dialog box for editing. If unchecked, the application does not lock assets when you open its **Properties** dialog box.


#### Clear manually locked tasks after edit

Unlocks a locked asset after you have edited its properties and clicked **OK** to create a new revision. If unchecked, the application does not remove the locks.

 **Note:** If you do not select either of the locking options, opening a change request will not lock it; you must manually lock and unlock it. If you select the **Exclusively Lock** option only, change requests that are not already locked become locked when you open them and unlocked when you click **Cancel** or **OK**. If you select the **Clear change request Locks** option only, any change request that you have locked manually becomes unlocked when you click **OK** to create a new revision. If you select both options, you can lock change requests manually or by opening them. These change requests become unlocked when you click **OK** to create new revisions or (if they were not locked prior to being opened) when you click **Cancel**.

## URL Options


<b>Display template</b>	Specifies a special template used to generate an HTML representation of an item when the item's URL is copied to the Clipboard. With no format, there is a default HTML representation that specifies the type of item and identifies it by name and number. When the text is generated from the template, the specified property values are substituted for the variables in <code>~~*~~</code> . The variables may be referenced by the same names used in report templates, as well as by the display name of the property. When using the display name, you can omit spaces, and case will be ignored. For example, if you use the following sample template for a file: <code>Task #~~Task Number~~:~~Status~~:~~Responsibility~~</code> , the HTML representation will be <code>Task #1,456:Ready To Start:Tom Smith</code> . This template is a super-set of that used by the Report feature of the client.
<b>Generate ID-based URLs</b>	Specifies the URL by ID rather than by name. For example, an ID-based URL would be <code>starteam://hostname:49201/12;ns=Project;scheme=id</code> , while a name-based URL would be <code>starteam://hostname:49201/myproject</code> .

 **Note:** Folders always use an ID-based URL.

## Topic Options

Window > Preferences > Team > StarTeam > Topic

Use the **Topic Options** to specify the criteria that the application uses to determine whether a topic has been read. You can also indicate how often the application should search for new topics and how topic locking issues should be handled.

<b>Mark as read</b>	<p><b>When topic is selected</b> Marks an asset as read as soon as you select it. An unread asset is displayed with bold font. A read asset is displayed with regular font.</p> <p><b>When selected for ___ seconds</b> Marks an asset read after it has been selected for the specified number of seconds. The range is from 15 to 9999 seconds.</p> <p><b>Only when manually marked as read</b> Marks a selected topic read when you choose <b>topic &gt; Mark as Read</b>.</p>
	 <b>Note:</b> Assets are always marked as read when you display their properties.
<b>System tray notifications</b>	<b>Check for new or modified topics</b> Checks for new or modified assets at regular intervals and lets you know that you have become responsible for new assets. If this option is checked, you must also specify the number of minutes



between checks in the Interval option below. When unchecked, the application does not place an icon in the system tray for a new change request.

**Interval (in minutes)** Specifies the number of minutes between automatic checks for new or modified assets. The default is 10 minutes.

## Locking

**Exclusively lock topic during edit** Locks an asset when you open its **Properties** dialog box for editing. If unchecked, the application does not lock assets when you open its **Properties** dialog box.

**Clear manually locked topics after edit** Unlocks a locked asset after you have edited its properties and clicked **OK** to create a new revision. If unchecked, the application does not remove the locks.



**Note:** If you do not select either of the locking options, opening a change request will not lock it; you must manually lock and unlock it. If you select the **Exclusively Lock** option only, change requests that are not already locked become locked when you open them and unlocked when you click **Cancel** or **OK**. If you select the **Clear change request Locks** option only, any change request that you have locked manually becomes unlocked when you click **OK** to create a new revision. If you select both options, you can lock change requests manually or by opening them. These change requests become unlocked when you click **OK** to create new revisions or (if they were not locked prior to being opened) when you click **Cancel**.

## URL Options

**Display template** Specifies a special template used to generate an HTML representation of an item when the item's URL is copied to the Clipboard. With no format, there is a default HTML representation that specifies the type of item and identifies it by name and number. When the text is generated from the template, the specified property values are substituted for the variables in `~*~`. The variables may be referenced by the same names used in report templates, as well as by the display name of the property. When using the display name, you can omit spaces, and case will be ignored. For example, if you use the following sample template for a topic: `Topic # ~*~: ~*~`, the HTML representation will be `Topic #34,132: Topic Title, Status - Active`. This template is a super-set of that used by the Report feature of the client.

**Generate ID-based URLs** Specifies the URL by ID rather than by name. For example, an ID-based URL would be `starteam://hostname:49201/12;ns=Project;scheme=id`, while a name-based URL would be `starteam://hostname:49201/myproject`.



**Note:** Folders always use an ID-based URL.

## Workspace Options

**Window > Preferences > Team > StarTeam > Workspace**

The **Workspace** personal options allow you to select a variety of options that affect the way your workspace operates.

**Confirm Delete** Displays a **Confirm** dialog box for deletions.

	<b>Move/Share</b>	Displays a <b>Confirm</b> dialog box for move and share operations.
	<b>Warnings</b>	Displays a <b>Confirm</b> dialog box for warnings.
<b>Display</b>	<b>Toolbars</b>	Displays <b>Toolbars</b> .
	<b>Status bar</b>	Displays the <b>Status Bar</b> .
	<b>Custom tools</b>	Displays custom tools created as part of StarTeam Extensions. If no custom tools are configured, clear the <b>Custom tools</b> check box to prevent custom tools from attempting to load with each view window.
	<b>Show history times as UTC</b>	Displays time stamps in the Time column of the history pane in UTC times. UTC times end in "Z" to differentiate them from local times (Z stands for the "zero meridian", which goes through Greenwich, England). Displays time stamps in local time when unchecked.
	<b>Sort view labels by name</b>	Automatically sorts view labels alphabetically.
<b>General</b>	<b>Restore folder selection on tab change</b>	Returns StarTeam to the last folder that was selected for a specific component tab. If you have not selected a particular tab in this session, StarTeam automatically selects the root folder for the view.
	<b>Reset scope to local on folder change</b>	Resets the <b>All Descendants</b> button on the toolbar every time you change a folder. This saves you the time StarTeam would take to scan items. If unchecked, you must select the button manually.
	<b>Maintain group state on folder/ scope change</b>	Keeps your place in the upper pane even when you change folders or reset the <b>All Descendants</b> button. If unchecked, all groups collapse when you make a folder or scope change.
	<b>Open URL in new window</b>	The default setting for the way a view displays in your workspace when you open a URL link. When this check box is selected, any StarTeam view you open via a URL opens in a new window.
	<b>Automatic refresh with maximum delay of _____ minutes</b>	Specifies the maximum number of minutes between refreshes of your project. This refresh is the equivalent of pressing <b>Shift + F5</b> and updates the <b>Folder Tree</b> and the upper pane. It happens every X minutes unless an operation by the user such as pressing <b>Shift+F5</b> forces a refresh and resets the timer.  Automatic refresh is designed to perform even if the client is minimized. When <b>Automatic refresh</b> enabled, if the network is down, <b>Automatic refresh</b> will attempt to refresh the data in the client resulting in a connection error.
	<b>Restore shortcuts at startup</b>	Reopen at startup any views that were left open when you exited the application the previous time.
	<b>Sort open windows by name</b>	If checked, sorts the opened windows by name. By default, windows are sorted by the order in which they are opened.
<b>Reports</b>	<b>Report output path</b>	Specifies the name and path to which your reports should be sent
<b>StarTeam client log</b>	<b>Log output path</b>	Specifies the name and path for your StarTeam.Log files. The StarTeam.Log file contains data about operations sent from your workstation to one or more servers, depending on what project views

you have open. The data includes the name of the project so that you can isolate data for a particular server when necessary.

### Log errors

Records errors that occur while you are using the client application. The errors log lists the date and time you started your server configuration and any errors or failed operations between the server and client. The application identifies each failed operation by an internal ID and provides an explanation. For example, you might see: `...Operation 40956 failed: TCP/IP Socket Error 10054:...` If you are logging both errors and operations, the application also logs the operation that the server was performing at the time of the error.

### Log operations and events that take at least \_\_\_\_\_ (milliseconds)

Specifies that StarTeam should record file operations and/or events that take at least the specified number of milliseconds, and should send them to the `.log` files. The milliseconds time setting stops the log from filling up with operations and events of little importance. The default, 10 milliseconds, is a reasonable setting.

The **Summary** option includes a breakdown of the time spent on the client and the server for each operation, and the **Details** option lists the server commands along with the summary.

This option records information on the date, time, and UI Operation number for each command executed by your workstation. Operations can be executed on either the server or the client.

### Log events

Specifies that StarTeamMPX events should be sent to the `.log` files. The log identifies the time and date on which a StarTeamMPX event (an automatic refresh or file status update) took place. The log prefaces a StarTeamMPX event as `Statistics for Events` and uses internal IDs and brief explanations to identify the server event.

The following example describes a status change for a file: `...Statistics for Events /1b21dd1-e208-51ea-01b2-1dd1e20851ea/Object/File/ Modify.`

You can log StarTeamMPX events only if you check **Enable MPX** on the **MPX** tab in the **Personal Options** dialog box. For StarTeamMPX related operations, any changes you make on the **Workspace** tab do not apply to projects already open. However, the application will log StarTeamMPX events for any projects you open after checking this option.

## Folders

This topic provides an overview of planning a folder tree and provides some tips for working with new and existing folders. The project or server administrator usually creates projects and project views. If you are a typical user, you routinely open a particular project view and manage your folders and their contents, such as files and change requests. Managing application folders is very similar to managing a project. You can create folders, delete folders, and modify their properties if you have the correct access rights.

## Folders and Items

The topics in this section provide information about how StarTeam manages folders and items and how to use them.

## Overview of Folders and Paths

In StarTeam, three types of folders play important, yet dissimilar, roles.

### Original workstation folder

Users set up this folder and its contents on a workstation, then use the **New Project Wizard** to create a new project. This folder, which may contain files and other folders, becomes the root folder of the new project – that is, it becomes the root folder of the project's initial or root view. StarTeam creates the project, the root view of the project, and root folder at the same time. The project, view, and root folder initially have the same name, although the name can be changed later.

### StarTeam folder

StarTeam uses the folders to group items in a project view. For example, a folder named `Source Code` can group source code files, requested changes to those files, and other related items. These folders can be created automatically at the same time as a project or added later by administrators or team members with the appropriate privileges. The hierarchy of folders in the current view appears as a folder tree in the project view window.

### Working folder on the workstation

A working folder is actually a property of a StarTeam folder, but is quite different, as it is an object controlled by the operating system. It stores files that are copied or checked out from StarTeam or that will be added to StarTeam. A folder is an object controlled from within StarTeam. Data about it is stored in the database that holds all project data.

A project, its root view, and its root folder all have the same working folder. If additional views of the project are created, each view and its root folder have the same working folder. The working folder for the root folder always has an absolute path, which starts with the drive letter and lists the appropriate directories until it reaches the working folder itself.

You can add to a project at any time after it has been created. These folders can be moved from or shared with another StarTeam view or added from a workstation.

## Understanding Working Folders

Understanding the relationship between application folders and their working folders is important because the working folder stores the files that you check in and check out.

Each folder has a default working folder from which you modify working files. For team members that use the same folders, the working folder structure on one person's workstation is often the same as those on another person's workstation.

When you check out a file, the application copies the requested file revision to the appropriate working folder. If the working folder does not already exist on your workstation, the application automatically creates it for you as you check out files that go in that folder.

The application expects you to add and check in new file revisions from those working folders. If the working folder does not exist on your workstation, you can create it manually or automatically using the **Create Working Folders** command. After the working folder exists, you can add files to it.

The exact location of a working folder is displayed as one of the application folder's properties.

## Alternate Working Folders

The view's working folder may not be the optimal choice for all users. You, or any other user with the access rights to do so, can select a more useful location for the view's working folder on your own workstation by designating an alternate working folder. For example, you might want to use a shorter path or a different drive letter. Remember that a working folder must point to a physically discrete location, such as a drive on your workstation or a personal directory on a shared file server. We do not recommend putting your settings on a mapped network drive.

The alternate working folder path for the view is specific to the workstation and user. For example, if you log onto the project as another user or use another workstation, your alternate working folder setting is not known.

When you designate an alternate working folder for the view, the path to the working folder for each child folder in the view may be similarly modified for your workstation.

For every folder in the hierarchy whose working folder is relative to the path of the view's working folder (as opposed to having an absolute path or an alternate working folder path of its own), your alternate path for the view's working folder becomes part of the paths to its child folders' working folders.


## Folder Paths

StarTeam often stores working folder paths from development environment applications as relative paths. For example, `.. \sc` may be the working folder for a project's `Source Code` folder. If you move a folder to another location in the hierarchy, its working folder may end up in an unexpected location. This result occurs because the application applies the relative path to the working folder path for the new parent folder. Therefore, if you move a folder, you may want to specify a working folder path that is not relative, to avoid accidentally changing the working folder path on users' workstations.


## Folder Tab


StarTeam includes a component tab for folders called **Folder**. When selected, this tab displays a main menu item and context -menu that contains many of the same menu commands that you would use when working with files, change requests, requirements, and so on. It is possible to perform some operations on multiple selected **Folder** items, such as adding files to a view.


Folder states are represented by the following folder icons:

 Regular folder.

 Invisible Folder: Indicates a folder where the **Visible** property has been unchecked in the **Folder Properties** dialog box.

 Not-in-View Folder: Indicates a folder on your local disk that does not map to a folder.

 Missing Working Folder: Indicates that local working folders do not exist.

 Folder uses an alternate working folder path than the default one set up by the project.

## Folders

The project or server administrator usually creates projects and project views. If you are a typical user, you routinely open a particular project view and manage *your* folders and their contents, such as files and change requests. Managing application folders is very similar to managing a project. You can create folders, delete folders, and modify their properties—if you have the correct access rights.

## Folder Hierarchy

When you create projects, you typically select locations on your workstations as the working folders for those projects. The working folder designated for a project also becomes the working folder for the project's root view and for the root folder in that view's folder hierarchy.

StarTeam treats folders as both containers and items. You can group items within a project view by placing them into folders. For example, a folder named `Source Code` can contain source code files and requested changes to those files. You can create folders automatically when you create a project, or add folders after you create the project. Project or server administrators (or team leads – this all depends on your organization) usually create projects, but anyone can create projects if they have the correct access rights. See your server administrator if you have questions regarding the access rights assigned to you.

When you create a project, StarTeam automatically creates the parent or root folder for that project at the same time. It is actually the root folder of the project's root (or initial) view. The project, view, and this root folder initially have the same name (although those names can be changed).

Usually, the user who creates a project sets up a hierarchy of folders on a workstation before creating the project. The user designates the root folder of that hierarchy as the project's working folder. Then the application can automatically create an application folder for each of the child folders in the hierarchy. The child folder becomes the application folder's working folder.

If child application folders are created at the time the project is created, then:

- The application folders' working folders were part of an existing hierarchy on the project creator's workstation.
- Their names are the same as the names of their working folders, but they can be changed later.
- Their working folders remain hierarchically connected to the root folder's working folder. That is, if you change the path to the root folder's working folder, you also change the path to this folder (unless you manually set an absolute path for these working folders). In other words, the application stores a relative path to each child folder.

One of the most important properties to notice about your folder is its working folder. You will need to know where on your workstation the application will copy file revisions that you check out so that you locate those revisions as needed for modifications. A number of other operations can be performed on folders, such as moving a folder or changing its branching behavior.

A working folder is a property of the folder and represents the actual location on your workstation where StarTeam saves files that you check-out. Despite the fact that these are both called folders, the working folder and the folder are not identical. Their differentiating characteristics include:

- The path to the working folder can be totally different from the path within the application to the application folder.
- An application folder is an object controlled from within the application. The data associated with this folder is stored in the database that stores all the project data.
- A working folder is an object controlled by your operating system. It stores files that are checked out from the application.

A project, its root view, and the root folder of the root view all have the same working folder. For additional views, each view and its root folder have the same working folder.

The working folder for the view/root folder always has an absolute path (starting with the drive letter and specifically naming the folders at subsequent levels until you reach the working folder itself).

If you look at the properties for the root folder, you will see that the working folder is the same. However, it is displayed in the **Complete Working Folder Path** display box instead of the **Default** field. Since you can only change the working folder at the view level, all of the fields for the root folder's working folder are always disabled.

For the child folders that were created at the same time as the project, the application stores the path to each working folder as a relative path.

### **Understanding Default and Alternate Working Folders**

Make sure that everyone is logged off from the StarTeam Server and that the StarTeam Server is locked before you change the Default Working Folder. It is just as critical to perform these actions as it is when you change custom fields or do anything else that affects all users.

When a view is created, a default location is specified for its working folder. If you change the Default Working Folder, not only the path to the working folder but the path to each child folder in the view may be similarly modified – not just for you, but for everyone working with that view. Therefore, before making such changes, it is important to understand the relationship of the working folder to the StarTeam view.

<b>Default Working Folder path</b>	Used by everyone sharing that view, unless they have specified an Alternate Working Folder path. Only change the Default Working Folder if you want to change the path for everyone who shares the view.
<b>Alternate Working Folder path</b>	Lets you specify a different location for your own working folder than the Default Working Folder. If you do not want to use the Default Working Folder path, specify an Alternate Working Folder path. Do not change the Default Working Folder. If you specify an Alternate Working Folder path, it is used instead of the Default Working Folder path. The Default Working Folder path must point to a location that is physically discrete for each user, such as a drive on that user's workstation or a personal directory on a shared file server.

The working folder for the view's root folder has an absolute path (for example `C:\New Product`). The path used for the working folder of a child folder depends upon how the child folder was created and what changes have been made to the path since that time. Generally, the working folder for a child folder is relative to that of the view (that is, relative to the working folder used for the root folder). For example, suppose that the path to the view's working folder is `C:\New Product` and that the root folder has a child folder named `Online Help`. In this case, the path to the `Online Help` working folder would be `C:\New Product\Online Help`. When the path to the view's working folder changes, the path to the child's working folder changes automatically.

If a new child folder is added to the view after it is created, the path to the child's working folder will usually be relative. However, if its working folder is on a different drive than the working folder for the root, its path will be absolute.

## Granting Folder-Level Access Rights

Setting access rights at the folder level is usually done when you want to allow certain groups (but not other groups) to access a particular branch of the folder hierarchy. For example, you may want only the **Writers** group to be able to access the branch that has `User Manual` as its root folder.

Setting access rights at the folder and the item levels has more consequences than setting rights at higher levels. When a child view is derived from a parent view, as all reference and most branching views are, it initially contains objects that belong to its parent. In branching views, these objects can branch into new objects that exist only in the child view. Just as a new view has no view-level access rights, folders and items that branch into new objects initially have no access rights at the folder or item level.

<b>This Folder and Child Folder Nodes</b>	The folder level has two nodes: <b>This Folder</b> , for the selected folder, and <b>Child Folders</b> , for the other folders in the folder hierarchy of the branch. This feature allows you to set different access rights for each node.
---	---

In the client, the root folder of a view can be indistinguishable from that view. If the view is the root (or initial) view of a project, the root folder can be indistinguishable from that project.

Using the **This Folder** node to set access rights for the root folder can therefore affect a user's access to a view. If the view is the root view, it can also affect the user's access to the project. Therefore, most administrators avoid setting folder-level access rights on a root folder, as these rights may interfere with view-level or project-level rights.

For example, suppose the **Developers** group is not granted the right to see the `User Manual` folder and that this folder is the root of a reference view. Then members of the **Developers** group cannot open that view, even if view-level access rights allow them to see the view. An error message appears when they try to open the view. Users who can see a project but not its root view also see an error message.

<b>Access Rights of Child Views</b>	If a child view includes child folders that have access rights in the parent view, its access rights depend upon whether it is a reference view or a branching view.
-------------------------------------	--

### Access Rights in a Reference View

The access rights in a reference view at the folder level are not independent of the access rights at the folder level in the parent view, as no branching will ever occur. You can see these access rights from either view if you have the rights to do so.

If you change access rights in the reference view, you simultaneously change the access rights in the parent view (and vice versa) because the folder in the reference view is the same object as the folder in the parent view.

### Access Rights in a Branching View

If the child view is a branching view, the access rights in the child view at the folder level are independent of the access rights at the folder level in the parent view, but only after the folder in the branching view actually branches.

Initially, any folder you see in the branching view is the same object that exists in the parent view. Therefore, it has the same access rights in both views. Initially, you can change access rights in the parent view (and vice versa), because the folder in the branching view is the same object as the folder in the parent view. Once the folder branches, however, a new object for that folder is created in the branching view. This object begins a life cycle of its own and no longer has any access rights at the folder level.



**Note:** Remember that branching a folder does not branch any of the folder's contents. Each item in the folder is treated separately.

The behavior of folders in a branching view affects the access rights:

- If a folder branches on change and you change one of its properties, its revision number changes. When the folder branches, it becomes a new object in the repository and no longer has any access rights at the folder level.
- If a folder does not branch on change and you change one of its properties, the revision number changes, but no new object is created. In this case, the folder retains its access rights in both views. Because both views still contain the same object, changes you make to the folder's access rights in one view also change that folder's access rights in the other view.

### Folder Access Rights

When you select the **Folder Tree > Advanced > Access Rights** command to display the **Folder Access Rights** dialog box, you see two folder nodes. The rights available from **This Folder** node apply to the selected folder only. The rights available from the **Child Folders** node apply to all the child folders of the selected folder. The dialog and following table refer to the current folder. The table describes the access rights that are available from the **This Folder** node in the **Folder Access Rights** dialog box.

**Note:** Because **This Folder** has no **Generic item container** subcategory for access rights, container rights for **This Folder** are on its **Child Folders** node. If **This Folder** is the root folder, these rights are set on the **Child Folders** node of the **View Access Rights** dialog box.

### Generic object rights

#### See item and its properties

View this folder's **Name**, **Exclude**, and **Files** tabs, which become available when **Folder > Properties** is selected. The **History** tab is controlled by the `See folder history` access right. The **Link** tab is controlled by the `See folder links` access right.

#### Modify properties

Change folder properties on the folder **Name** and **Exclude** tabs. Properties include folder name, description, use of inherited and local exclude lists, and contents of the local exclude list. If the folder is not a root folder, the working folder and alternate working folder settings are also properties. For root folders, the working folders are view properties and not controlled by this access right.



<b>Delete from folder</b>	Delete this folder from its parent folder. Be aware that if you can delete any of this folder's parent folders, you can still delete this folder.
<b>Change item access rights</b>	Change the access rights for this folder. If you change this setting, be sure that you remain one of the users who can change access rights.
<b>See history</b>	See this folder's <b>History</b> tab, which is available when <b>Folder &gt; Properties</b> is selected.
<b>Perform maintenance</b>	Change the revision comments for past revisions.
<b>Set exclusive locks</b>	Lock folders exclusively.
<b>Break exclusive locks</b>	Remove someone else's exclusive lock on the folders.

### Label Rights

<b>Attach/Adjust view labels</b>	Add a view label to this folder. Move a view label from one revision of this folder to another. This right controls direct manipulation of labels for this folder at the folder level. It does not stop users from attaching a view label to this folder when a view label is created.
<b>Detach view labels</b>	Remove a view label from this folder. Be aware that if users can delete view labels, they can detach a view label from this folder by deleting the view label from the view, regardless of the setting for this right.
<b>Attach/Adjust revision labels</b>	Add a revision label to this folder. Move a revision label from one revision of this folder to another. This right controls direct manipulation of revision labels for this folder at the folder level.
<b>Detach revision labels</b>	Remove a revision label from this folder. Be aware that if users can delete revision labels, they can detach a revision label from this folder by deleting the revision label from the view, regardless of the setting for this right.

### Link Rights

<b>See links</b>	See the links involving this folder.
<b>Create links</b>	Link this folder to other folders and items.
<b>Modify links</b>	Change a link for this folder.
<b>Delete links</b>	Delete a link for this folder.

### *Child Folder Access Rights*

When you select the **Child Folders** node from the **Folder Access Rights** dialog box, the available rights apply to the child folders of the selected folder. The **Child Folders** node is also available from the **View Access Rights** dialog box and the **Project Access Rights** dialog box. In these cases, the rights apply to all child folders in the current view or all the child folders in the project, respectively.

Below is a description of the access rights available from the **Child Folders** nodes in the **Project Access Rights**, **View Access Rights**, or **Folder Access Rights** dialog boxes.

## Generic item rights

<b>See item and its properties</b>	See the selected folder's child folders or the selected project's or view's folders in the folder hierarchy in the left pane on the screen. You can also view the <b>Name</b> and <b>Exclude Properties</b> dialogs, which open when <b>Folder &gt; Properties</b> is selected. The <b>History</b> tab is controlled by the <b>See folder history</b> access right.
<b>Modify properties</b>	Change folder properties on the <b>Name</b> and <b>Exclude</b> tabs for child folders. The properties include the folder's name, description, use of inherited and local exclude lists, and the contents of the local exclude list. If a child folder is not a root folder, the working folder and alternate working folder settings are folder properties. If it is the root folder, the working folders are view properties and not controlled by this access right.
<b>Delete from folder</b>	Delete the selected folder's child folders or the selected project's or view's folders from their parent folders. Be aware that if you can delete any of this folder's parent folders, you can still delete this folder.
<b>Change item access rights</b>	Change the access rights for the selected folder's child folders or the selected project's or view's folders. If you change this setting, be sure that you remain one of the users who can change access rights.
<b>See history</b>	See the <b>History</b> tab, which is available when <b>Folder &gt; Properties</b> is selected. This action applies to the selected folder's child folders or the selected project's or view's folders.
<b>Perform maintenance</b>	Change the revision comments for past revisions.
<b>Set exclusive locks</b>	Lock child folders exclusively.
<b>Break exclusive locks</b>	Remove someone else's exclusive lock on the child folders.

## Label rights

<b>Attach/ Adjust view labels</b>	Add a view label to the selected folder's child folders or the selected project's or view's folders. Move a view label from one revision of a child folder to another. This right controls direct manipulation of view labels for child folders at the folder level. It does not stop users from attaching a view label to child folders when a view label is created.
<b>Detach view labels</b>	Remove a view label from the selected folder's child folders or the selected project's or view's folders. Be aware that if users can delete view labels, they can detach a view label from child folders by deleting the view label from the view, regardless of the setting of this right.
<b>Attach/ Adjust revision labels</b>	Add a revision label to the selected folder's child folders or the selected project's or view's folders. Move a revision label from one revision of a child folder to another. This right controls direct manipulation of revision labels for child folders at the folder level.
<b>Detach revision labels</b>	Remove a revision label from the selected folder's child folders or the selected project's or view's folders. Be aware that if users can delete revision labels, they can detach a revision label from this folder by deleting the revision label from the view, regardless of the setting of this right.

## Link rights

- See links** See the links involving the selected folder's child folders or the selected project's or view's folders.
- Create links** Link the selected folder's child folders or the selected project's or view's folders to other folders and items.
- Modify links** Change a link for the selected folder's child folders or the selected project's or view's folders.
- Delete links** Delete a link for the selected folder's child folders or the selected project's or view's folders.

## Generic item container rights

- Create and place in folder** Create a folder in a parent folder, view, or project in which the **Child Folder Access Rights** dialog box has this option.
- Share/Move out of folder** Share or move a folder in a parent folder, view, or project if its Child Folder Access Rights dialog has this option. Be aware that the access rights set for that folder and its contents, along with any rights set for specific child folders and items within that branch of the folder hierarchy, accompany the folder into the new folder.
- Change behavior or configuration** Change the branching ability and configuration of folders that reside in a parent folder, view, or project if its **Child Folder Access Rights** dialog box has this option.

## Granting Item-Level Access Rights

Although access rights can be set on individual items, this is rarely done. For example, if you really need to allow only one person to know about a particular file, you can give only that person access rights to that file. However, by default, the owner of the file and anyone belonging to a group with the correct privileges can still see that file.

To ensure that only that one person can access the file, you would have to stop the StarTeam Server from checking for privileges. Then the access to every object would be controlled solely by access rights.

Like folders, items in a child view retain the access rights they had in the parent view until they branch into new objects. Items lose their access rights only when branching.

- Moving Folders or Items** When you move a folder or an item, the access rights set for it at the folder or item level go with it. It also has the same behavior, which either allows it or stops it from branching on change.
- Sharing Folders or Items** When you share a folder or item, the access rights set for it at the folder or item level accompany it, until the folder or item branches.
- When you share a folder or item, its behavior may change. When shared, the behavior immediately becomes able to branch on change, even if the **Branch On Change** check box was disabled in the original location. Whether the **Branch On Change** check box is selected or cleared depends on the property setting for the destination view **Set Items That Are Shared Into View To Branch On Change**.
- When the folder or item branches in its new location, a new object is created in the repository, and that new object initially has no access rights at the folder or item level.

## Generic Item Access Rights

The following table describes the access rights that are available from the **File** nodes in the **Project Access Rights**, **View Access Rights**, **Folder Access Rights**, and **File Access Rights** dialog boxes.

### Generic item rights

<b>See item and its properties</b>	See files in the files list (upper pane) and view file properties by selecting <b>File &gt; Properties</b> .
<b>Modify properties</b>	Change the file properties. Modifiable properties include the archive/file name, description, executable bit setting (useful only for non-Microsoft Windows platforms), compression, storage options, and custom properties. If used, an alternate property editor (APE) may restrict the properties that can be modified and the users who can modify them still further.
<b>Delete from folder</b>	Delete files from their folders.
<b>Change item access rights</b>	Change access rights for the files. If you change this setting, be sure that you remain one of the users who can change access rights.
<b>See history</b>	See file history in the history pane.
<b>Perform maintenance</b>	Change the revision comments for past revisions.
<b>Set exclusive locks</b>	Lock files exclusively.
<b>Break exclusive locks</b>	Remove someone else's exclusive lock on the files.

### Folder Properties

This topic presents the folder properties and their descriptions as displayed in the **Folder Properties** dialog box. The **Folder Properties** dialog box contains the following tabbed pages of properties.

#### Name tab

The following properties are on the **Name** tab.

<b>Name</b>	Displays the name of the file.
<b>Description</b>	Displays the file description.
<b>Created By</b>	Displays the name of the person who created the file.
<b>(Created) On</b>	Displays the date on which the file was created.
<b>Visible</b>	Indicates whether the file is to be visible in the view or not.
<b>Working Folder</b>	Displays the path to the default and alternate working folders. <b>Default</b> This path location points to the default working folder, and applies to everyone accessing the project repository. <b>DO NOT CHANGE</b> this path unless you are a project administrator. <b>Alternate</b> This path points to an alternate working folder on your machine. Specifying an alternate working folder affects only you, not any of the other team members.
<b>Complete Working Folder Path</b>	Displays the complete path to the selected working folder (default or alternate.)

## Exclude tab

The following properties are on the **Exclude** tab.

**Files To Be Excluded** Indicates which files or types of files to exclude from visibility in the folder. The exclude list has no effect on files that are already part of the project. It only affects those with **Not In View** as their status. Exclude lists can be inherited from parent folders. **Exclude** options include:

**Inherit And Use Local Exclude List** Indicates that files matching the exclude list specifications set for this folder and those of its parent folder will be excluded.

**Use Local Exclude List** Indicates that files matching the only specifications set for the exclude list of this folder will be excluded.

**No Exclude List** Specifies that all files are included in the folder.

**Local Exclude List** Displays the exclude specifications to use for excluding files from this folder. The exclude list is limited to a maximum of 255 characters. It contains file specifications (using the standard \* and ? wild cards), separated by commas, spaces, or semicolons. To include a comma, space, or semicolon as part of the specification, enclose the specification in double-quotes. For example,

```
*.exe, *.dll p*z.doc;*.t?t "test *.*"
```

**Inherited Exclude List** Displays the exclude specifications to use from the parent folder for excluding files from this folder.

## History tab

The **History** tab displays all the revisions of the folder. The following properties are displayed on the **History** page for each revision.

**View** Displays the name of the view to which this folder belongs.

**Revision** Displays the file revision number.

**Modified By** Displays the name of the person who created the folder.

**Modified Time** Displays the date and time the revision created.

**Comment** Displays a comment explaining why the revision was created.

**Dot Notation** Displays the branch number of the revision.

## Link tab

The following properties are on the **Link** tab which displays all the links to this folder.

**Created By** Displays the name of the person who created the link to the folder.

**Created On** Displays the date on which the link was created.

**View** Displays the name of the current view if the link was created in the current view, or displays the name of view where the link was created and from which the link is shared.

**Folder** Displays the name of the folder in which the folder or item in the link resides.

**Item Type** Identifies the type of item to which the target end of the link is attached. This item is listed in the link list.

<b>Item</b>	Identifies the item to which the target end of the link is attached. It is identified by its folder name, file name, change request number, task number, topic number, or requirement number.
<b>Item Details</b>	Describes the item, using a folder description, file description, change request synopsis, task name, topic title, or requirement name.
<b>Item Version</b>	Displays the version number of the target end of the link if that revision is in the current view. When no revision number is displayed in the column, that end of the link is floating rather than pinned.
<b>Selection Version</b>	Displays the version number of the source end of the link if that revision is in the current view. When no revision number is displayed in the column, that end of the link is floating rather than pinned.
<b>Comment</b>	Displays a comment about this particular link.
<b>File Status</b>	Displays the status of a file that is linked to the folder.
<b>Locked By</b>	Displays the name of the person who locked the file linked to the folder.
<b>Folder Path</b>	Shows folder path information only when the linked item is in the same view. Otherwise, it displays the message, "Unavailable. Item in another view."

## Folder Fields

This section lists all the folder fields in alphabetical order.



**Note:** Client-calculated fields cannot be used in custom email notifications or StarTeam Notification Agent. Reports can use any field name.

### Branch On Change (Advanced)

Values: No, Yes.

Internal Identifier: BranchOnChange.

Indicates whether the item will branch when it changes.

The value is No if the item's behavior is not set to **Branch On Change**. Reasons for this may be:

- The item is in the root or a reference view and the **Branch On Change** feature is disabled.
- The item is in a branching view but has already branched as a result of a change, which, in turn, results in the **Branch On Change** feature becoming disabled.
- The item is in a branching view, but its behavior currently does not permit it to branch on change. This means that modifications are checked into the parent view.



**Note:** If the value is No, the value of the **Branch State** explains the No.

### Branch State (Advanced)


Values: Branched, Not Branched, Root.

Internal Identifier: BranchState.

Indicates whether an item has branched in the child view, is still unbranched (and therefore is part of the parent view), or was created in the view in which it resides.

The values Branched and Not Branched apply to items in branching views. The value Root applies to items created in the view in which the item currently resides.

If the view is a reference view, it reflects the state of the item in the reference view's parent.

<b>Comment</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>Comment</code>.</p> <p>The initial 2000 characters provided as the reason for changing an item's properties or contents are stored in the <b>Short Comment</b> field. The <b>Comment</b> field stores those 2000 characters and any additional text. Changing an item's properties causes the application to create a new revision.</p> <p> <b>Note:</b> To include a <b>Link</b> comment, the <b>Comment</b> field is the value to use in an HTML report.</p>
<b>CommentID (Advanced)</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>CommentID</code>.</p> <p>The ID number assigned to the revision comment. Displays -1 if no revision comment was supplied.</p>
<b>Configuration Time</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>ConfigurationTime</code>.</p> <p>Indicates the time to which an item is configured. If you configure an item to a specific time, this field contains that time. If you configure an item to a label or promotion state, this field shows either the time at which the label was created, or the time at which the label associated with the promotion state was created.</p>
<b>Created By</b>	<p>Values: list of users, <code>&lt;None&gt;</code>.</p> <p>Internal Identifier: <code>CreatedUserID</code>.</p> <p>The name of the user who created the first revision in the view. This is either the user who added the item to the project, or the user who checked in the revision that branched.</p>
<b>Created Time</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>CreatedTime</code></p> <p>The time at which the first revision in the view was created.</p>
<b>Creating Project</b>	<p>Values:</p> <p>Internal Identifier: <code>CreatingProject</code></p>
<b>Deleted By</b>	<p>Values: list of users, <code>&lt;None&gt;</code>.</p> <p>Internal Identifier: <code>DeletedUserID</code>.</p> <p>The name of the user who deleted the item. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.</p>
<b>Deleted Time</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>DeletedTime</code>.</p> <p>The time at which an item was deleted. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.</p>
<b>Description</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>Description</code>.</p>

The description provided for an item at the time it was added to the view, including any later edits to it.

**Dot Notation**

Values: `text`.

Internal Identifier: `DotNotation`.

The branch revision number, for example, `1.2.1.0`.

**Dot Notation ID (Advanced)**

Values: `number`.

Internal Identifier: `DotNotationID`.

The ID assigned to a particular branch revision number. For example, if a folder was added to the current view (as opposed to inherited by the current view), its branch revision number is `1.x` and its branch revision ID is `0`. If a folder was branched in the current view, its branch revision ID is dependent on the revision number in the parent view and the number of IDs already assigned in the current view. For example, if a folder's revision number in the parent view is `1.7` at the time of the branch, and another folder with that same parent revision number was given the Branch Revision ID `6`, this folder will also be given the Branch Revision ID `6`.

**End Modified Time (Advanced)**

Values: `date/time`.

Internal Identifier: `EndModifiedTime`.

The date and time at which a revision ceased to be the tip revision. Although this field can be displayed in the upper pane, its value is always blank. This is because, at any given configuration time, the item is still the tip revision.

**Exclude Flags**

Values: `Inherit` and `Local Exclude List,Local Exclude List,No Exclude List`.

Internal Identifier: `ExcludeFlags`.

The flag which specifies the types of file to be excluded from the folder.

**Exclude Spec**

Values: `text`.

Internal Identifier: `ExcludeSpec`.

The file specification (using the standard `*` and `?` wild cards), separated by commas, spaces or semicolons. To include a comma, space, or semicolon as part of the specification, enclose the specification in double quotes.

**Folder Path**

Values: `text`.

Internal Identifier: `Folder Path` (contains spaces).

The path to the folder. This is not the path to the working folder.

**Is Action Overridden?**

Values: `No`, `Yes`.

Internal Identifier: `Is Action Overridden?` (contains spaces).

**Local Path**

Values: `text`.

Internal Identifier: `Local Path` (contains spaces).

The local path to the folder. This is the path to the working folder.

**Locked By**

Values: list of users, `<None>`.

Internal Identifier: `ExclusiveLocker`.



	The name of the user who has exclusively locked a folder.
<b>Modified By</b>	<p>Values: list of users, &lt;None&gt;.</p> <p>Internal Identifier: ModifiedUserID.</p> <p>The name of the user who last modified the item.</p>
<b>Modified Time</b>	<p>Values: date/time.</p> <p>Internal Identifier: ModifiedTime.</p> <p>The time at which an item was last modified. The item may have been checked in or had its properties changed. For folders, this has nothing to do with the working folder. Use <b>Local Time Stamp</b> for the time a working folder was last modified.</p>
<b>Name</b>	<p>Values: text.</p> <p>Internal Identifier: Name.</p> <p>Displays the name of the item.</p>
<b>Non-Exclusive Lockers</b>	<p>Values: text.</p> <p>Internal Identifier: NonExclusiveLockers.</p> <p>The names of the users who have locked the folder non-exclusively.</p>
<b>Object ID</b>	<p>Values: number.</p> <p>Internal Identifier: ID.</p> <p>Each item is assigned an object ID when it is added to a view. For applicable items, when it is branched in a child view, it is assigned another object ID. The original ID belongs to the folder in the parent view.</p>
<b>Parent Branch Revision (Advanced)</b>	<p>Values: number.</p> <p>Internal Identifier: ParentRevision.</p> <p>The last digit in the branch revision number before an item branched. For example, if this number is 7, the branch revision was 1.7 at the time the item branched (becoming 1.7.1.0, as seen in the item's history). This number is -1 if an item was not inherited from the parent view.</p>
<b>Parent ID (Advanced)</b>	<p>Values: number.</p> <p>Internal Identifier: ParentID.</p> <p>The object ID of an item in the parent view. The Parent ID is -1 if this view has no parent view.</p>
<b>Parent Revision (Advanced)</b>	<p>Values: number.</p> <p>Internal Identifier: PathRevision.</p> <p>The revision number at which an item branched. For example, if this number is 8, this item's revision number in the parent view was 8 at the time the item branched. The history should show that revision 9 in the first revision in the current view. This number is 0 if this item was not inherited from the parent view.</p>
<b>Read Only (Advanced)</b>	<p>Values: No, Yes.</p> <p>Internal Identifier: ReadOnly.</p>

Indicates whether the item's configuration is read-only (as in a rollback configuration of a view)/its behavior does not allow it to branch on modification. For folders, do not confuse a read-only configuration (an application issue) with a read-only folder (an operating system issue). A read-only folder cannot be edited and saved to disk. A folder whose configuration is read-only can be edited and saved to disk; it just cannot be checked in.

**Revision Flags  
(Advanced)**

Values: 0.

Internal Identifier: `RevisionFlags`.

Internal use only.

**Root Object ID  
(Advanced)**

Values: number.

Internal Identifier: `RootObjectID`.

The object ID of the oldest ancestor of an item. For example, if an item was not inherited from a parent view, the root object ID is the same as its object ID. If it was inherited from a parent view, the root object ID is the Parent ID, or the item's Parent ID.

**Share State**

Values: `DerivedShare`, `Not Shared`, `Root Share`.

Internal Identifier: `ShareState`

Indicates whether this item is shared. `Not Shared` means that the item is not shared. `Root Share` means that the item is shared and this item is the original (or root) reference. `DerivedShare` means that the item is shared, but this item is not the original (or root) reference.

**Short Comment**

Values: text.

Internal Identifier: `ShortComment`.

Stores the initial 2000 characters provided as the reason for changing an item's properties or contents. Additional text is stored in the **Comment** field.

**Status**

Values: `Current`, `Deleted on Disk`, `Deleted on Server`, `Merge`, `Missing`, `Modified`, `Not In View`, `Out Of Date`, `Unknown`.

Internal Identifier: `Status`.

Indicates the relationship between the copy of an item in your working folder and the tip revision in the repository.

**Version  
(Advanced)**

Values: number.

Internal Identifier: `RevisionNumber`.

The last number in the branch revision number. For example, if the branch revision number is 1.3.1.2, the version is 2.

**View**

Values: list of views, `<None>`.

Internal Identifier: `ViewID`.

The name of the view in which the item last branched. For example, if an item is inherited from a parent view but is branched in a child view, the value of this field in the child view changes from the name of the parent view to the name of the child view for the revision that branched and subsequent revisions in the child view.

**\*\*\*Working Folder** Values: number.  
 Internal Identifier: WorkingFolder.  
 The name of the working folder.

## Provider Properties

This topic presents the StarTeam **Provider** properties and their descriptions as displayed in the **Provider Properties** dialog box when opened from the context menu. The information in the **Provider Properties** dialog box is different for each type of selected, shared resource: project, folder, or file.

### Shared Project

The following properties are displayed for the shared project.

**Server Description** Displays the name of the StarTeam Server on which the shared project is located.

**Project Name** Displays the name of the selected shared project.

**View Name** Displays the name of the view which contains the shared project.

**Folder** Displays the folder on the StarTeam Server that is used for sharing.

**Change Sharing** Opens the **Share a Project** dialog box where you can change the sharing options.

**Change View** Opens the **Select a StarTeam View** dialog box where you can change which view you are sharing.

**Change Kind** Reveals the types and directions of changes that can be found in any resource within the project. If it reads **[in sync]**, there are no files and folders that are not "current".

**Base Bytes** Shows both the dot-notation and the ID for the **StarTeam** folder used for sharing.

**Remote Bytes** Shows the same as Base Bytes, but reflects the current situation on the StarTeam Server. It is possible for the dot-notation to be higher, indicating a newer version is available in the repository.

### Shared Folder

The following properties are displayed for the shared folder.

**Server Description** Displays the name of the StarTeam Server on which the shared project is located.

**Project Name** Displays the name of the selected shared project.

**View Name** Displays the name of the view which contains the shared project.

**Folder** Displays the folder on the StarTeam Server that is used for sharing.

**Change Kind** Reveals the types and directions of changes that can be found in any resource within the project. If it reads **[in sync]**, there are no files and folders that are not "current".

**Base Bytes** Shows both the dot-notation and the ID for the StarTeam folder used for sharing.


**Remote Bytes** Shows the same as **Base Bytes**, but reflects the current situation on the StarTeam Server. It is possible for the dot-notation to be higher, indicating a newer version is available in the repository.

**Convert Change Kind To** Allows you to invert the direction and type of a deletion or an addition. This is sometimes necessary when a project was re-shared or hooked up with another view. Changes on the StarTeam Server or in the workspace may be interpreted assuming the wrong intention. A folder listed as **[outgoing deletion]** may simply have been

added to the view by another developer at some time, and thus be considered an **[incoming addition]**. The two buttons, **Incoming Deletion** and **Outgoing Addition**, allow you to change

## Shared File

The following properties are displayed for the shared file.

<b>Server Description</b>	Displays the name of the StarTeam Server on which the shared project is located.	
<b>Project Name</b>	Displays the name of the selected shared project.	
<b>View Name</b>	Displays the name of the view which contains the shared project.	
<b>Folder</b>	Displays the folder on the StarTeam Server that is used for sharing.	
<b>MD5:</b>	Displays the current MD5 hash value for the local revision.	
	<b>Stored Remote MD5:</b>	Displays what the provider remembers the MD5 hash value to have been at the last check in or check out.
	<b>Current Remote MD5:</b>	Displays the MD5 hash value for the tip revision. By comparing the stored MD5 hash value with the current MD5 hash value, the provider can detect incoming changes in spite of differing remote and local MD5 values due to EOL differences.
<b>Base Modification Timestamp:</b>	Shows the last date and time the local and remote files were current.	
	<b>Local Modification Timestamp</b>	Shows the date and time of the last modification to the local file.
	<b>Remote Modification Timestamp:</b>	Shows the date and time of the last modification to the remote file.
<b>Change Kind</b>	Reveals the types and directions of changes that can be found for any file resource within the project. If it reads <b>[in sync]</b> , there are no files and folders that are not "current".	
<b>Base Bytes</b>	Shows both the dot-notation and the ID for the StarTeam file used for sharing.	
<b>Remote Bytes</b>	Shows the same as <b>Base Bytes</b> , but reflects the current situation on the StarTeam Server (tip revision). It is possible for the dot-notation to be higher, indicating a newer version is available in the repository.	
<b>Convert Change Kind To</b>	Allows you to invert the direction and type of a deletion or an addition. This is sometimes necessary when a project was re-shared or hooked up with another view. Changes on the StarTeam Server or in the workspace may be interpreted assuming the wrong intention. A folder listed as <b>[outgoing deletion]</b> may simply have been added to the view by another developer at some time, and thus be considered an <b>[incoming addition]</b> . The two buttons, <b>Incoming Deletion</b> and <b>Outgoing Addition</b> , allow you to change.	
<b>Stored EOL Conversion</b>	Specifies which EOL (End of Line) conversion settings to store. When you click <b>OK</b> or <b>Apply</b> after changing this setting, the MD5 values will be recomputed and the status refreshed in an attempt to identify the type of change.	
	 <b>Note:</b> StarTeam uses MD5 hash values exclusively for status computation, therefore it needs to remember which EOL conversion was in effect at the last check in or check out so it can adjust the computed MD5.	


# Working with Folders and Items

This section contains information related to working with folders and items.


## Creating a Working Folder

In StarTeam, each of the child folders in the view has its own working folder, which is generally relative to the path of the root working folder. When you check out a file, StarTeam copies the requested file revision to the appropriate working folder. If the working folder does not currently exist on your workstation, StarTeam automatically creates it for you when you check out files.

You can also add new files to StarTeam from a working folder. If the appropriate folder does not yet exist on your workstation, you can create it automatically by using the **Create Working Folders** command. Once the working folder exists, you can place files in it and add them to StarTeam.

 **Important:** The default working folder must point to a location that is physically discrete for each user, such as a drive on that user's workstation or a personal directory on a shared file server.

1. Select a folder in the **Server Explorer** or the **Eclipse Explorer**.
2. Right-click on the folder, and choose **Team > Create Working Folders**. After the working folder exists, you can copy files to it or create files in it and add them to the StarTeam repository.

 **Note:** If the working folder path for a shared or moved folder exceeds the operating system's maximum working folder path length of 254 characters (including [ \ ] backslashes), StarTeam will not allow you to create the working folder and displays an error message.

## Adding Folders to Views

1. Open a StarTeam perspective or view.
2. Do one of the following:
  - Choose **StarTeam > Folder > New**.
  - In the toolbar, click **Create a new StarTeam Folder**.
  - Open the Server Explorer, select a StarTeam project and/or view, and choose **New > Folder**.

The **New Folder** wizard opens.

3. Select a project and view clicking **Next** after each selection
4. Select a parent folder for the new folder in the folder tree.
5. Click **Next**. The new folder will be created as a child of the selected folder, and the **Folder Name** page of the wizard displays.
6. Type a name and description for the new folder.  
The folder name can be up to 254 characters.
7. Do one of the following:

Option	Description
<b>Select Default</b>	This option automatically provides a working folder by appending the name of the new StarTeam folder to the path of the working folder.
<b>Select Change Default</b>	Type in or browse to the path for an existing working folder. When you browse for a path, you create an absolute path to this folder's working folder.
<b>Select Map to Workspace</b>	This creates the working folder in your Eclipse-defined workspace path for the associated Eclipse project.

The Sub-folder Inclusions (for including child folders) page of the **New Folder** wizard displays the new folder and any child folders. If the working folder has child folders, a StarTeam folder is created for each of them.

8. Do one of the following:

- To exclude a child folder from your project, clear the check box next to the folder name in the folder tree.
- To exclude all child folders, click **Deselect All**.
- To reselect folders you have excluded, click **Select All**.

9. Click **Finish**.

## Attaching New Labels to Folders

Determine whether a label is a revision label or a view label by opening the **Label** view, and double-click the label (or select the label, and then choose **Properties** from the context menu). A revision label has a name and a description. A view label has a name, description, and a configuration time.

1. Select a folder in the Server Explorer or from one of the Eclipse Explorers.

2. Choose **StarTeam > Folder > Labels**.

*Alternative 1:* From the Server Explorer, right-click on the folder, and choose **Labels**. *Alternative 2:* From one of the Eclipse Explorers, choose **Team > Labels**.



**Note:** If you open the **Properties** dialog box using the **Team > Labels** command, expand **StarTeam Provider**, and select **Labels**.

The **Properties** dialog box opens.

3. Select the revision that will receive the new label.

4. Click **New**. The **Attach a New Revision Label** dialog box opens.

5. Type a name and description for the label in the appropriate text boxes.

6. *Optional:* Check **Frozen** to ensure that only the selected revision can have this label.

7. Select one of the following options:

- **Folder Only** to attach a label to only the selected folder.
- **Folder and Items Contained in Folder** to attach a label to the folder and its items.
- **Everything in Subtree Rooted at Folder** to attach a label to the folder, its items, and its child folders and their items.

8. Click **OK**. The **Properties** dialog box displays the newly-created revision label.

9. Click **OK**.

## Changing a View's Default and Alternate Working Folders

Make sure that everyone is logged off from the server and that the server is locked before you change the *Default Working Folder*. It is just as critical to perform these actions as it is when you change custom fields or do anything else that affects all users.

When you change the **Default Working Folder**, not only the path to the working folder but the path to each child folder in the view may be similarly modified—not just for you, but for everyone working with that view.



**Caution:** Do not change the **Default Working Folder** unless you are a project administrator. These default settings affect all users and incorrect settings cause other users to be unable to check out StarTeam files. The default settings should only be set to the name of the folder. If you want to use a different location for your working folder than the **Default Working Folder** path, specify an **Alternate Working Folder** path.



**Important:** Changing the view's working folder for working Eclipse projects is not recommended. Eclipse projects should be recreated after changing the view's working folder. If you want to use a different location for your working folder other than the **Default Working Folder** path, it is recommended that you specify an **Alternate Working Folder** path.

1. Choose **StarTeam > View > Properties** to open the **Properties** dialog box.

2. Select **View**.
3. Click the **Name** tab.
4. Do one of the following:
  - Select **Alternate** to create a different working folder for only yourself.
  - If you are a project administrator, select **Default** to specify the default repository path for all users.
5. Type the name of a new working folder or browse for a path to a working folder. If you browse for the path, it becomes an absolute path. This path can be edited, however, to enable you to work on a computer that uses a different letter for its hard drive.



**Note:** It is important that the **Default Working Folder** point to a location that is physically discrete for each user, such as a drive on that user's workstation or a personal directory on a shared file server.

## Changing Name or Description of Folders and Items

1. Select a folder or item in one of the StarTeam or Eclipse views.
2. Right-click the selected item and choose **Properties**.

If you open the **Properties** dialog box from one of the Eclipse Explorers, expand **StarTeam Provider** in the dialog box to view the StarTeam properties. If you open the **Properties** dialog box within a StarTeam view or from the Server Explorer, this action is not required.
3. Change the **Name/Description** for the folder or item and click **OK**. The folder or item name/description is changed in both the StarTeam repository and in your working folder.

## Compare Revisions

You can use the **Compare With** context menu commands for, among other things, comparing local versions of files and folders to those in the StarTeam Server repository.

Using the **Compare With** context menu commands, you can complete the following actions:

- Compare your local file to the latest tip revision stored in the repository.
- Compare a local folder or file to a labeled version stored in the repository.
- Compare a local folder or file to a copy in a different StarTeam view stored in the repository.
- Compare a local folder or file to a copy of that same item determined by a date.
- Compare a local file to a specific revision stored in the repository.
- Select two files or two folders and compare their properties.

### Comparing Local Files to Tip Revisions

1. Select a file in one of the Eclipse Explorers.
2. Right-click and choose **Compare With > Latest from StarTeam**.

The comparison editor opens displaying the differences between the two file versions.

### Comparing to a Label

1. Select a folder or file in one of the Eclipse Explorers.
2. Right-click on the item and choose **Compare With > Label**.

For folders, the **Synchronize** view opens along with the **Compare** editor. For files, the **Compare** editor opens. In the **Compare Editor**, you can select a Label and the editor automatically updates to reflect the differences between the local copy and the labeled copy.

## Comparing to a Different View

1. Select a folder or file in one of the Eclipse Explorers.
2. Right-click on the item and choose **Compare With > View**.
3. Select the appropriate view, and click **OK**.

For files, the **Compare** editor opens displaying the differences between the local and remote file. For folders, the **Synchronize** view opens.

## Comparing by Date

1. Select a folder or file in one of the Eclipse Explorers.
2. Right-click on the item and choose **Compare With > Date**.
3. Select the desired date and time.
4. Click **OK**.

For files, the **Compare** editor opens displaying the differences between the local and remote file. For folders, the **Synchronize** view opens.

## Comparing a Local File to a Revision

1. Select the file in one of the Eclipse Explorers.
2. Right-click on the file and choose **Compare With > Revision**.

The **Compare** editor opens. The available revisions display at the top of the **Compare** editor. Selecting a revision displays the differences between the local file and the selected revision in the lower section of the **Compare** editor.

## Comparing Properties for Files or Folders

1. Select two files or two folders in one of the Eclipse Explorers.
2. Right-click on one of the selected items and choose **Compare With > Compare Properties**. The informational **Compare Properties** dialog box opens. By default, differing properties display in blue text.
3. Click **Close**.

## Configuring (Rolling Back) Folders and Items

You can configure (or roll back) an individual folder or item to a specific view label, promotion state, or date and time. Essentially, all rollbacks are made to a particular date and time. For example, if you roll back to a view label, you essentially roll back to the revision of the folder or item that existed on the date and time at which that label was attached. Unlike a view, a folder or item retains its roll-back configuration until you manually change it or until the folder or item branches. When you close the view, the folder or item does not immediately return to its current configuration.

Rolling back a folder does not re-configure any of the items or child folders associated with it. It only rolls back the folder properties to the values they had at the configuration time. Depending on the folder behavior, the folder may become read-only, in which case its properties cannot be changed.

The configuration of a folder affects the new items or child folders that can float into it. For example, in a floating branch view, you can keep items from floating into a particular folder by configuring the folder to a particular label, promotion state, or point in the past. Later, you can re-configure the folder to floating so that it can receive new items from its parent. However, the items added to the parent while the folder was not floating will never automatically go into the folder. They must be manually shared. To freeze a folder or item at a certain point in time so that it cannot be changed:

- Change its configuration to a point in the past.



- Make sure that its branching behavior is either disabled or not set to **Branch on Change**.



**Caution:** There is no way to locate folders that have been configured to a point in the past unless you make a note of them. Use this feature with caution.

## Rolling Back Folders

1. Select the folder in the Server Explorer or the Eclipse Explorers that contains the task that you want to modify.
2. Open a StarTeam perspective or view, and choose **StarTeam > Folder > Advanced > Behavior**.  
*Alternative1:* If selecting a folder from the Server Explorer, right-click and choose **Advanced > Behavior** from the context menu. *Alternative2:* If selecting a folder from an Eclipse Explorer, right-click and choose **Team > Advanced > Behavior**.  
The **Item Behavior** dialog box opens.
3. On the **Configuration** tab, select a configuration option:
  - **Labeled Revision:** This option uses the folder or item revision with the specified view label as the tip revision. Existing view labels are listed in reverse chronological order based on the time for which they were created. This option is disabled if the view has no labels defined.
  - **Promotion State Configuration:** This option uses the folder or item revision with the view label assigned to the selected promotion state as the tip revision. This option is disabled if this view has no promotion states defined.
  - **Configuration As Of:** This option uses the folder or item revision just prior to the specified point in time as the tip revision. It defaults to the current date and time, but you can select a date and time in the past, as long as it is after the time when the folder or item was created.
4. Click **OK**.

## Modifying Folder Configurations

1. Select the folder in the Server Explorer or the Eclipse Explorers that contains the task that you want to modify.
2. Open the **Item Behavior** dialog box, in one of the following ways:
  - If using the Server Explorer, right-click on the folder, and choose **Advanced > Behavior**.
  - If using one of the Eclipse Explorers, right-click on the folder, and choose **Team > Advanced > Behavior**.
3. Check **Branch on change** in the **Modify** tab.  
This is not available on the main branch of the item.
4. On the **Configuration** tab, choose one of the following:
  - **Current configuration** radio button to keep all revisions current.
  - **Labeled configuration** radio button to point the configuration to a specific label.
  - **Promotion state configuration** radio button to point to a specific promotion state.
  - **Configuration as of <date>** radio button to point to a specific date.
5. Check **Apply to entire folder tree** to have all items beneath the selection reflect the same configuration settings.

## Deleting Folders

1. Open the Server Explorer.
2. Select the folder you want to delete in the project view.
3. Right-click the selected folder, and choose **Delete**. A dialog box opens prompting you to confirm the deletion.

4. Click **OK**.

## Displaying Location References

1. Open the Reference view.
2. Click **Link with Selection**.
3. Do one of the following:
  - Select a folder in the Server Explorer or from one of the Eclipse Explorers.
  - Select an item from one of the StarTeam views.

The Reference view automatically updates showing any references for the selected folder or item.

## Emailing Item Properties

You can send a text representation of selected items (except files) as an email message, along with additional text. The information sent for each item includes the fields displayed in the upper pane. For items such as change requests, the item's properties, which are the same as its contents, are sent in the email. For files, only the properties can be sent. However, a shortcut to the item can be included.

Items are considered to have been sent by the application, not by you. Therefore, you may want to copy yourself on the email. Otherwise, you will not receive the message.



**Note:** If you set up a filter and email an item, only the fields displayed by the filter are sent to the recipient.

1. Do one of the following:
  - Select a file or folder from one of the Eclipse Explorers, and choose **Team > Send To** from the context menu.
  - Open one of the StarTeam views, and choose **Send To** from the context menu.
2. Click **To** or **CC** to open a dialog box for selecting the primary or secondary email recipients.

Select the email recipients by moving the team member names from the **Available Users** to **Selected Users** list and click **OK**.

3. Type a **Subject**.
4. Optionally, check **Send A Copy To Myself** if you want to receive a copy of the email.
5. Optionally, check **Attach Item Shortcut** to include a shortcut to this specific item in the email.
6. Type any additional information in the **Add Text To The Mail Message** text box.
7. Click **Send Now** to send the message.



**Note:** Unlike automatic email notification, this message will not display the word “notification” in the subject line. Do not confuse email messages sent by individuals with email notification messages automatically sent by the StarTeam Server. If your administrator has enabled email notification, you will automatically receive email messages notifying you about items for which you are responsible and topics for which you are listed as a recipient.

## Finding Items

You can search all items displayed in the upper pane for the data contained in any displayed field. For example, you can locate a change request by its number or search for a file with a particular name, status, time stamp, or size.

1. Select a folder in Server Explorer or in one of the Eclipse Explorers.
2. Enter **CTRL+F**. The **Find** dialog box opens.
3. Type part or all of the data in the **Search For** field.



**Note:** Wild cards are not supported.

4. Select **Forward** to search the upper pane from the top to the bottom, or select **Backward** to search the upper pane from the bottom to the top.
5. Select **Starting at: Currently Selected Item** to begin searching from the item that is presently selected, or select **Starting at: First Item** to search from the first item in the upper pane.
6. Select either **All Displayed Fields** or **This Field**. If you select **This field**, select the field for which you want to search from the list.
7. Check **Match Case** if a case-sensitive search is appropriate.
8. Click **Find** to search.



**Tip:** Use **F3** to find the next item that matches the search text and **Shift+F3** to find the previous item that matches the search text.

## Hiding Folders and Files

1. Select the folder in the Server Explorer or the Eclipse Explorers that contains the item that you want to hide.
2. Click **StarTeam > Folder > Properties** from the main menu. The **Properties** dialog box opens.
3. Select the **Name** tab and uncheck the **Visible** option. This hides the folder and the files it contains
4. Click **OK**.



**Note:** To make the folder visible again, check the **Visible** option in the **Properties** dialog box.

## Highlighting Items of Interest

To highlight specific items (except audit entries) on the upper pane, you can add a flag to them. For example, you might wish to flag items related to a particular customer request.

Flags are set, viewed, and removed by the user who created them. If an item has been flagged, the **Flag** field displays **Yes**. If an item is not flagged, the **Flag** field displays **No**.

1. Select the item you want to flag.
2. Do one of the following:
  - If selecting an item from one of the Eclipse Explorers, right-click on the item, and choose **Team > Flag**.
  - If selecting an item from the Change Request, Requirement, Task, or Topic views, right-click on the item, and choose **Flag**.
  -
3. If reviewing flagged items in the Change Request, Requirement, Task, or Topic view, click the **Filter** list for the view, and choose the **Flagged Items** filter.

## Linking Items Internally or Externally

This procedure describes how to link two items, either internally in the same server configuration, or linking between two items located on different server configurations, called *external linking*.

In StarTeam, an *item* is a file, change request, requirement, task, or topic. A link is a connection between two folders, two items, or a folder and an item on the same server, or on two different servers (called *External Links*).

Creating links can be quite useful. For example, linking a file to a change request allows you to mark it as fixed when you check in the edited file. By linking files to the requirements document that the files fulfill, you can easily refer to or update the document.

You can create several links at the same time if you want to link several items of the same type to one particular item. For example, you might wish to link several change requests to a single file. To accomplish this, you can create links using the **Folder Tree** menu, component menu, context menu, or **Link** button on the toolbar.



**Note:** When creating external links between items on different server configurations, both server configurations need to be opened in Eclipse to be able to create or view the external links.

1. Open a StarTeam perspective. If you want to link two items on different server configurations, open both server configurations and perspectives.
2. Select file(s) or folder(s) in the Server Explorer or in one of the Eclipse Explorers, or by selecting the item(s) in one of the Change Request, Requirement, Task, or Topic views.
3. Click **Create/Complete Link** in the main toolbar.
4. Select the folder or item(s) for the end of the link in the project on the other StarTeam Server. This can be:
  - A StarTeam folder (if you have not already selected a folder).
  - One or more other files.
  - One or more change requests or change packages.
  - One or more requirements.
  - One or more topics/responses.
  - One or more tasks/subtasks.

To locate all items, you may need to switch to a different component tab or use the **All Descendants** button on the toolbar.

5. To complete the link, click **Create/Complete Link** again in the main toolbar.

You can also view a link by selecting either of its ends. The end you select, whether a folder or an item, is called the source. The other end of the link is called the target and is listed in the **Item Type** column on the **Link** pane.



**Note:** External links can also be created using drag-drop. With both views open, select the source item, press **CTRL + SHIFT**, then drag-drop it on the target item.

## Locking and Unlocking Items

Before changing the contents of a file or editing item properties, you should exclusively lock the file or item. This action informs other team members that you intend to make changes. Files, change requests, requirements, tasks, and topics can all be locked.

Exclusively locking an item prevents others from creating new revisions of it before the lock has been released. You can lock and unlock any type of item as a separate operation. In addition, you can lock and unlock files as part of the check-in and check-out processes.

If an item is exclusively locked by someone else, you can review its properties but cannot change them. Normally the words **Read Only** and the name of the user who has locked the item will appear on the title bar.

### Locking an Item Using the Toolbar

1. Select one or more items.
2. Click the **Lock** button on the toolbar. The selected items become exclusively locked, and you are listed as the user who has locked them.



**Tip:** To unlock an item, select the locked item and click **Unlock** on the toolbar.

## Locking an Item Using the Menu

1. Open the appropriate view to select a folder, file, change request, task, topic, or requirement.
2. Right-click an item and choose **Lock/Unlock**. This opens the **Set My Lock Status** dialog box.
3. Select a lock status option:

<b>Unlocked</b>	Removes your exclusive or non-exclusive lock on the selected items.
<b>Exclusive</b>	Prevents others from creating new revision of this item (until you release the lock or another person breaks your lock).
<b>Non-exclusive</b>	Indicates that you are working on the item and may possibly make changes (not recommended for items other than files).

4. Optionally, check **Break Existing Lock** to break another team member's lock on the item.

If e-mail is enabled, StarTeam will send an e-mail message to the team member whose lock has been broken to inform him or her of this fact.



**Note:** You must be granted the appropriate privileges to be able to break another person's locks.



**Tip:** To unlock an item, select the locked item and click **Unlock** on the toolbar.

## Marking Items Read or Unread

Change requests, requirements, tasks, and topics that you have not read display in boldface type if you are the user responsible for the item, or if you are the recipient of the topic. After you have reviewed the item properties, the boldface type is replaced by regular type.

To reread all the items on a certain subject, it is sometimes convenient to mark all of the items unread to ensure that you do not miss any. You can also mark an entire requirement, task, or topic tree as unread.

1. Open the Change Request, Requirement, Task, or Topic view.
2. Select one or more items that you want to make bold (unread) or not bold (read).
3. Right-click the selected items and choose **Mark As Unread** or **Mark As Read**.

The items are marked as specified, and the type style changes accordingly.

## Moving Folders or Items

Folders and items, such as files (including Not in View files) and change requests, can be moved from one project view to another as long as the two views are on the same server configuration. Moving a folder also moves its contents, its child folders, and their contents. When an item is moved to another project view, it belongs to the new view, although its behavior, configuration, and other properties do not change. It loses any labels it had in the previous view, however, because labels cannot move from view to view. Also, if you roll back the view to an earlier point in time, you will no longer see the folders and/or items that have been moved.

Moving a folder or item within a view causes that folder or item to be copied in that view's child or parent views, if branching has not occurred. In this application, a move is a copy operation followed by a delete operation, and delete operations are not propagated from view to view for folders and items that have not branched. Therefore, the view in which the move was made has one copy of the folder or item in the new location, while the related views have two copies of the folder or item, one in the original location and one in the new location, the equivalent of a share.



**Note:** When you move a folder or an item, the access rights set at the folder or item level accompany it. Also, in some cases, moving a folder or item to another view enables its disabled **Branch on Change** check box.

## Moving a Folder or Item Within the Same View

1. Open the Server Explorer.
2. Drag the folder or item that is to be moved from one location in the view to another.



**Tip:** You can optionally move the working folder/file using this dialog box. This option is automatically selected. If you do not want to move the working folder or file, clear the check box.

3. A message box appears asking you to confirm.
4. Click **Yes**.

## Moving a Folder or Item Between Two Different Views

1. Open the Server Explorer and open two project views.
2. Make sure that both windows are open.
3. Drag the folder or item from one view to the other.
4. A message box appears asking you to confirm.
5. Click **Yes**.

## Opening a Local Folder

The following procedures describe how to open a local folder in a file browser so you can perform basic file and folder management tasks.

### Opening a Local Folder from a Folder Selection

1. Select up to two folders in the Server Explorer or one of the Eclipse Explorers.
2. Right-click the selected folder and choose **Open Local Folder**.

This opens a **Windows Explorer** for each location on disk that corresponds to a selected folder. This applies to all folders except those whose status is **Missing** since their local folders do not exist.

## Replacing Files and Folders

Describes how to replace files and folders with other versions in the StarTeam repository.

### Rolling Back a Folder or File

1. Select the folder or file in one of the Eclipse Explorers.
2. Right-click on the file, and choose **Replace With > Latest from StarTeam**.

### Replacing a Folder/File with a Specific View Configuration

1. Select the folder or file in one of the Eclipse Explorers.
2. Right-click on the file, and choose **Replace With > View**.
3. In the resulting dialog box, select a StarTeam view.
4. Click **OK**.

### Replacing a Folder/File Under a Specific Labels

1. Select the folder or file in one of the Eclipse Explorers.
2. Right-click on the file, and choose **Replace With > Label**.
3. In the resulting dialog box, select a desired labeled configuration.
4. Click **OK**.

## Rolling Back a Folder/File to a Specific Date

1. Select the folder or file in one of the Eclipse Explorers.
2. Right-click on the file, and choose **Replace With > Date**.
3. Select a date and (optionally) time configuration that you wish to roll back to.
4. If including a *time* option, specify whether the time is local or UTC.
5. Click **OK**.

## Sorting and Grouping Data

1. Open the **Change Request, Requirement, Task, Topic, or Audit Log** view.
2. Choose **Filters > Sort and Group**. The **Sort and Group** dialog box opens displaying four group boxes, each indented slightly more to the right than the one above it. The first group box designates a primary sort order, the second designates a secondary sort, and so on.
3. *Optional:* Check the **Show Advanced Fields** check box to list all the fields in **First By** and **Then By** list boxes.
4. Select a field from the **First By** list box.  
If you are grouping the items, you do not need to display the field in the view. If you are not grouping the items, you can sort them based on a field that is not displayed, but you will not be able to tell where one group leaves off and the next begins.
5. Click **Ascending** or **Descending** option button.  
The default setting is ascending order.
6. Select **Group By** to group the items which have the same values in this field.  
If you do not select any additional sort options, text fields are sorted in ASCII order. Enumerated and user ID fields are sorted by their internal order or internal keys. That is, enumerated fields are sorted in the order given to them by the person who created the field; user ID fields are sorted in the order in which they were created. The application disables the **Sort Options** button for numeric and date/time fields.
7. *Optional:* Click **Sort Options** for additional sorting selections.  
The **Sort Options** dialog box opens.
  - a) Select **By internal key or order** to sort enumerated and user ID files by their internal order or internal keys.
  - b) Select **As text** to sort enumerated and user ID fields by the names of their possible values. For text fields, **As Text** is your only choice.
  - c) Uncheck **Case-sensitive** to sort alphabetically or check it to sort in ASCII order (where uppercase letters precede lowercase letters).
8. Add secondary and lower order sorts by using the **Then By Group** boxes as needed.

## Viewing or Modifying Item Properties

This section explains how to use the standard properties dialog to edit item properties. Depending on how your team has set up the application, you may see a totally different dialog box called an alternate property editor (APE).

Every time the properties of an item are modified, a new revision of that item is created. If you modify a property, you should also create a revision comment explaining the modification using the **Comment** tab provided in the **Properties** dialog box.

1. Select an item in one of the StarTeam or Eclipse views.
2. Do one of the following:
  - Select an item and double click on it.

- Right-click the folder or item and choose **Properties**.
3. Modify any of the property fields in the corresponding **Properties** dialog box that opens, then click **OK**.

## Comparing Properties

1. For files and folders, open the Server Explorer or one of the Eclipse Explorers.  
For other item types, open the **Change Request**, **Requirement**, **Task**, or **Topic** views.
2. Select two items in the open view.  
Use **Ctrl+Click** to select the second item.
3. For files and folders selected in one of the Eclipse views, right-click and choose **Compare > Compare Properties**.  
For items selected in the Server Explorer or in one of the item (change request, task, topic, and so on) views, right-click, and choose **Compare Properties**.  
The **Compare Properties** dialog box opens.
4. Click **Closed**.

## Setting Active Process Items

Setting an active process item is a convenient way of saving time when you know that you will be adding files or checking them in later.

You can make a selected change request, task, or requirement the active process item for the current view, an open view on the same server, or a different view on the same server.



**Note:** You can only specify one active process item for each view. Setting a second active process item for the same view at one time clears the first one.

1. Open your project in a StarTeam perspective.
2. Click the **Change Request**, **Requirement**, or **Task** tab and select the item you want to use as the active process item.
3. Do one of the following:
  - Right-click and choose **Set Active Process Item > Current View** to choose the current view.
  - Right-click and choose **Set Active Process Item > [view name]** to choose from the listed opened views on the server.
  - Right-click and choose **Set Active Process Item > Select View** to open a dialog box and choose any other view on the server.



**Note:** You can also set the currently-selected item as a process item by using the **Use As Active Process Item** button on the toolbar.

The active process item you selected is used by default when you add files or check them in. However, you can override this default and select another appropriate item when adding or checking in files.



**Tip:** After you finish with a process item, you should right-click it and choose **Clear Active Process Item** so that it cannot be accidentally reused. That removes the information from the status bar and keeps the process item from reappearing in the **File Add** or **File Checkin** dialog boxes.

## Selecting Items

1. Open the **Change Request**, **Requirement**, **Task**, or **Topic** views.
2. Click the toolbar list box in the corresponding view, and choose one of the following:
  - **Select > Select All:** Selects all displayed items in the view.



**Note:** The **Audit Log** view also provides this menu command.



- **Select > By Query:** Opens the **Select Query** dialog box where you can choose the desired query to apply for displaying in the view.
  - **Select > By Label:** Opens the **Select a Label** dialog box where you can specify a label for displaying items in the view.
3. To select items in Server Explorer, select an item, right-click and choose **Team > Show in Server Explorer**.

## Copying Items

You can copy items from one server and paste them to another server. This excludes Folders and Files. To copy a StarTeamItem from one Server and paste the item in a different Server:

1. Open a source view on one server and the target view on another server.
2. In the source view, navigate to the item to copy.
3. Right-click on the item and choose **Team > Copy**.
4. Navigate to the target view on the other server where you want to paste the item.
5. Right-click in the view and choose **Team > Paste**. A new Item is created on the selected Server with the same values as the copied Item. When the paste operation is selected, a confirmation dialog asks if you also would like to create an External Link between the two Items.

## Links: Internal and External

A link is a connection between two folders, two items, or a folder and an item on the same server, or on two different servers (called *External Links*). Creating links can be quite useful. For example, linking a file to a change request allows you to mark it as fixed when you check in the edited file. By linking files to the requirements document that the files fulfill, you can easily refer to or update the document.

In addition, linking files to change requests enables you to mark the change requests as fixed when you check in the corresponding files. In turn, if you link each set of files to the requirements document that the files fulfill, you can easily refer to or update the document.

A link does not provide a connection to a single share (or reference), but to all related shares and branches of an item. Links are not affected by any item operations, such as branching, moving, sharing, and so on. By default, a link connects the tip revisions of the linked pair.

Links can either be pinned or floating:

**Pinned** You can lock the link to the tip revision. The context menu in the link view enables you to pin links to the source or target items or both.

**Floating** You allow the link source or target to change from tip revision to tip revision as new revisions are created. The context menu in the link view enables you to float links to the source or target items or both.

Links, as with all other items, have context menus in their views which allow you access to more information about the item.

## Linking Items Internally or Externally

This procedure describes how to link two items, either internally in the same server configuration, or linking between two items located on different server configurations, called *external linking*.

In StarTeam, an *item* is a file, change request, requirement, task, or topic. A link is a connection between two folders, two items, or a folder and an item on the same server, or on two different servers (called *External Links*).

Creating links can be quite useful. For example, linking a file to a change request allows you to mark it as fixed when you check in the edited file. By linking files to the requirements document that the files fulfill, you can easily refer to or update the document.

You can create several links at the same time if you want to link several items of the same type to one particular item. For example, you might wish to link several change requests to a single file. To accomplish this, you can create links using the **Folder Tree** menu, component menu, context menu, or **Link** button on the toolbar.



**Note:** When creating external links between items on different server configurations, both server configurations need to be opened in Eclipse to be able to create or view the external links.

1. Open a StarTeam perspective. If you want to link two items on different server configurations, open both server configurations and perspectives.
2. Select file(s) or folder(s) in the Server Explorer or in one of the Eclipse Explorers, or by selecting the item(s) in one of the Change Request, Requirement, Task, or Topic views.
3. Click **Create/Complete Link** in the main toolbar.
4. Select the folder or item(s) for the end of the link in the project on the other StarTeam Server. This can be:
  - A StarTeam folder (if you have not already selected a folder).
  - One or more other files.
  - One or more change requests or change packages.
  - One or more requirements.
  - One or more topics/responses.
  - One or more tasks/subtasks.

To locate all items, you may need to switch to a different component tab or use the **All Descendants** button on the toolbar.

5. To complete the link, click **Create/Complete Link** again in the main toolbar.

You can also view a link by selecting either of its ends. The end you select, whether a folder or an item, is called the source. The other end of the link is called the target and is listed in the **Item Type** column on the **Link** pane.



**Note:** External links can also be created using drag-drop. With both views open, select the source item, press **CTRL + SHIFT**, then drag-drop it on the target item.

## Deleting Links

You can delete an existing link from either the source end or the target end of the link.

1. Open the **Link** view.
2. Click **Link with Selection**.
3. Select the folder or item that is the source or target of the link in one of the StarTeam views, Server Explorer, or Eclipse Explorers that links to a folder or another item.
4. Select one or more links to delete.
5. Right-click the selected links and choose **Links > Delete Link**.

## Linking Files to Process Items

If process rules are enforced for a project, linking and pinning new file revisions to a process item is required. Otherwise, this step is optional, and you can select any change request, requirement, or task as a process item.

## Linking and Pinning a File Revision to a Process Item

1. From one of the Eclipse Explorers, choose **Team > Check In** from the context menu. The **Check In** dialog box opens.
2. Expand **Process Item**, and check **Link and pin process item**.
3. Click **Select**. The **Select Process Item** dialog box opens.
4. Select one of the following to limit the list of possible process items:
  - List All Permitted Items** Displays all items that can be used as process items. If process rules are not enforced, the list contains all change requests, requirements, and tasks.
  - List All Permitted Items Assigned To Me** Displays all the items for which you are responsible that can be used as process items.
  - List Linked Items** Lists the process items that are already linked to at least one of the files you are checking in. No process item appears on the list more than once, even if it is linked to several files. Also, when a process item is linked to more than one file, the dialog box displays the name of only one file. Despite this fact, the application will update or create links for every file being added.
5. Select the **Change Request, Requirement, Task, or Custom Component** tab to restrict the list to a specific type of item.
6. Select one item in the list to be the active process item and click **OK**.
7. Optionally, in the **Check In** dialog box, check **Mark Selected Item As Fixed/Finished/Complete** if work on the process item is completed.
8. Complete filling in the fields in the **Check In** dialog box and click **OK**.



**Tip:** If process rules are enforced, the use of some change requests, requirements, or tasks as process items may not be permitted because of their status. If you select such an item and click **OK**, the application notifies you of this fact. By double-clicking the item in the list box, you can display its properties and change it to a permitted status for a linked process item.

## Linking and Pinning a File Revision to the Active Process Item

1. Click the **File** tab.
2. Select one or more files.
3. Choose **Team > Check In** from the context menu to display the **Check In** dialog box.
4. Check **Link and Pin Process Item**.
5. Optionally, in the **Check In** dialog box, check **Mark Selected Item As Fixed/Finished/Complete** if work on the process item is completed.



**Note:** Selecting this option will change the value of the process item's status property to the **Closed** state specified on the **Project Properties** dialog box.

6. Complete filling in the fields in the **Check In** dialog box and click **OK**.

## Linking Specific Revisions

Each end of a link has an associated start revision and an end revision that determines the range of revisions to which the link applies. The start revision is always fixed at the time of the creation of the link and is set to the first revision on the current branch. The end revision is under the user's control and may be fixed (or pinned), which puts an upper bound on the linked revisions, or floating, which does not. If a link end is pinned, it is always attached to the same version of the linked folder or file. If a link end floats, it moves from revision to revision, as new revisions of the linked folder or item are created.

By default, a link connects the tip revisions of the linked pair. The revisions selected for both links appear as columns on the **Link** pane.

Determining whether a link is visible on a given item is simple. If any of the revisions between the start and the end revision defined for the link are in the history of the selected item, it is visible. Otherwise, it is not.

## Linking to a Specific Revision

To link to a specific revision:

1. Open the **Link** view, and click **Link with Selection** in the Link view toolbar.
2. Select an item in one of the StarTeam views, Server Explorer, or Eclipse Explorers that links to a folder or another item. The Link view displays all links for the selected item.
3. Select a link.
4. Right-click the selected link and choose **Properties**. The **Properties** dialog box opens.
5. Optionally, type a description or comment about the link in the **Description** or **Comment** field. This text will appear in the **Comment** or **Description** column of the **Link** view.
6. Do one of the following in the **Source Item** group:
  - Click **Pin**: Displays the **Select Version** dialog box. Select a specific folder or item revision from the list. This revision number will appear in the **Selection Version** column of the **Link** view.
  - Click **Float**: The link is always connected to the tip revision of this item.
7. Do one of the following in the **Is Linked to Target Item** group:
  - Click **Pin**: Displays the **Item Version** dialog box. Select a specific folder or item revision from the list. This revision number will appear in the **Item Version** column of the **Link** view.
  - Click **Float**: The link is always connected to the tip revision of this item.
8. Click **OK**.



**Note:** You can link items from a project view on one server to an item in another project or view on a different server. This is called an external link.

## Customizing Link Item Properties

You can view or modify folder and item properties directly from the **Link** pane.

1. Click **Link with Selection** in the **Link** view's toolbar.
2. Select an item in one of the StarTeam views, Server Explorer, or Eclipse Explorers that links to a folder or another item.
3. Right-click a link and choose **Properties**. The **Properties** dialog box opens. It displays information about both the Source and Target Items.

## Customizing Link Properties

You can view or modify link properties from the **Link** view.

1. Click **Link with Selection** in the Link view toolbar.
2. Select an item in one of the StarTeam views, Server Explorer, or Eclipse Explorers that links to a folder or another item. The Link view displays all links for the selected item.
3. Right-click a link in the **Link** view and choose **Properties**.
4. The **Properties** dialog box opens where you can view or modify certain properties or add a description or comment.

# Finding Files Associated with Active Process Items

When you have files associated with an active process item, you can quickly find all associated file changes by following the steps below.

1. Open the view containing the active process item.

You can see what item is the active process item by looking at the Eclipse Status Bar.

The Status Bar displays the **Active Process Item** icon, followed by the name of the item.



**Tip:** The Server Explorer also reflects when you have an active process item specified at the view level on the **View** icon followed the active process item informational text.

2. Open the **Link** view.
3. Click **Link with Selection** in the toolbar.
4. Select the active process item in the appropriate **Change Request**, **Requirement**, or **Task** view.
5. Right click on a link and choose **Select Linked Item**. The view opens containing the target of the associated item.

# Reviewing Linked Change Requests

1. Select the file that is linked to the change request in one of the Eclipse Explorers.
2. Right-click on the file, and choose **Team > Check In**. The **Check In** dialog box opens.
3. Expand the **Change Requests** section. This section expands and displays a list of any **Change Requests Linked In This View**.



**Note:** No change request displays in the list more than once, even if it is linked to several of the files you are checking in. When a change request is linked to more than one file, the list displays the name of only one of the files.

# Change Requests

The *change request* component provides a defect tracking system that allows you to record defects in products, projects, or services and suggest possible enhancements. A change request is a request to change something within the scope of a project. For example, you might suggest a product enhancement or request a fix for an error or problem. To use the change request tracking system effectively, you need to understand the model on which it is based.

The change request component allows you to:

- Attach change requests to any folder. In the application, change requests can be attached to any project folder or shared among folders or other views in the same server configuration. You can also link a change request to any other item, such as a file. In many other defect tracking systems, a change request can be associated only with a project, even though it requires modification of a particular file.
- Save time when updating change requests. When you check in a file or group of files, you can indicate the change requests that are fixed by the files being checked in. This feature saves the time required to change the status of each change request separately.
- Make only appropriate status changes. When you create a change request, the status options are *New*, *Open*, *Deferred* or a resolution. The resolutions are *Cannot Reproduce*, *As Designed*, *Fixed*, *Documented*, and *Is Duplicate*. After resolution, a change request can only be verified or reopened. After verification, a change request can only be closed or reopened.
- Benefit from automatic changes based on the status of the change request. The application automatically changes the person responsible to coincide with the current status of the change request. When a change request is resolved, the responsibility for the change request automatically reverts to the person who entered the change request, who is usually the best person to verify its resolution.

When a change request is reopened after being resolved, the responsibility is automatically set to the user who resolved it. If desired, you can override these automatic changes and make another person responsible.

- Base change requests on the build in which the change request is resolved. When a change request receives a **Fixed** or **Documented** status, the value of its **Addressed In Build** field becomes **Next Build**. When that build label is created, the application replaces **Next Build** with the name of the build label, letting testers know the build to use when verifying change requests.



**Note:** This help system explains how to use the standard property dialog to create and edit change requests. Depending on how your team has set up the application, you may see a different dialog called an alternate property editor (APE). Even if you use the standard property dialog for change requests, your company or team leader may implement change request guidelines that differ from those discussed in this help system.

## Change Request View

The Change Request view provides a powerful search and/or reporting mechanism that allows change requests and suggestions about possible improvements to a project to be entered, assigned to a team member, and tracked.

You can open the **Change Request** view by taking one of the following actions:

- Select **Window > Show View > Change Request**
- Select **Window > Show View > Other**, and choose **Change Request** from the StarTeam tree node.

When you open the **Change Request** view, a list of change requests displays.

All change requests shown in the **Change Request** view have the following characteristics:

- Are attached to the folder selected from the Server Explorer.
- Match the filter selected from the **Filter** list box.
- Match the depth specified by the **All Descendants** button.

### Using the Change Request List

From the Change Request list that displays in the **Change Request** view, you can perform a variety of operations, including the following options:

- **View unread items:** To see the change requests for which you are responsible but have not reviewed, look for the change requests in boldface type. The change requests in regular type are those that you have read or those for which you are not responsible.
- **View properties:** To open the **Properties** dialog box for a specific change request, double click on it.

### Filter and Sort Change Request Data

You can sort the change requests by the data in a column and change the display based on a predefined filter available in the Change Request view toolbar.

You can sort change requests by clicking on a column header to sort the displayed change requests based on the value in that column. The sort is usually in ascending numeric or alphanumeric order, depending on the data. The ascending order for Severity displays from Low to Medium to High. The ascending order for Status reflects the life cycle of the change request, so these items display from New to Closed. The ascending order for Priority displays from No to Yes. To change the sort order from ascending to descending, click the header a second time. A triangle indicating the direction of the sort appears on the header of the primary sort column.

### Default Change Request View Columns

You can view the following default columns in the **Change Request** view:

- **CR Number:** Number that StarTeam assigned to the change request when it was first submitted.
- **Synopsis:** Short explanation of the change request.
- **Type:** Defect or Suggestion.
- **Status:** Indication of the change request life cycle. Status classifications include New, Open, Cannot Reproduce, As Designed, Fixed, Documented, Is Duplicate, or Deferred.
- **Severity:** Seriousness of the change request. Severity can be Minor, Normal, Major, or Critical.
- **Responsibility:** Person who is responsible for the change request.
- **Addressed In:** Build in which the change request is resolved.
- **Addressed By:** Name of user who resolved the change request. A change request has been resolved when its status becomes Cannot Reproduce, As Designed, Fixed, Documented, or Is Duplicate.
- **Last Build Tested:** Build in which the change request occurs.
- **Work Around:** Solution to a change request other than a fix.
- **Modified Time:** Time the change request was last modified.
- **Priority:** Indication that the change request is or is not a Priority.
- **Description:** Complete explanation of the defect or suggestion, including the steps to be taken to reproduce the problem.
- **Test Command:** Command that can be used to test the change request's solution.
- **Fix:** Solution to the problem addressed by the change request.
- **Entered On:** Date and time the change request was submitted.
- **Entered By:** Person who submitted the change request.

### Change Request View Toolbar and Context Menu Actions

In addition to the other capabilities described in this section, the Change Request view toolbar and context menu enable you to use the following options:

- Apply predefined filters for viewing lists of change requests.
- Search (CTRL+F) for change requests.
- Link (unlink) the view to the selected StarTeam item.
- Optionally show all descendants to determine the depth for which the client displays information
- Select all items.
- Select items using a label or pre-defined query.
- Determine which fields to display in the view.
- Perform up to fourth order sorts.
- Create, edit, copy, or delete queries
- Save or reset current view filters
- Create, edit, save, or delete filters
- Create new change requests
- Open change requests in various views, including History, Label, Link, and Reference views
- Modify and review change request properties
- Email a text representation of the change request's properties, along with additional text. The information sent includes the fields displayed in the view.
- Compare the properties of two change requests.
- Set the lock status.
- Mark the item as read or unread.
- Add, edit, enable, or disable custom fields.
- Create URLs and HTML representations and copy them to the Windows Clipboard and paste them into email applications, Word documents, Excel spreadsheets, and so on to quickly open the item in StarTeam
- Create, attach, and detach labels.
- Flag and remove flags from files.
- Delete change requests.

- Create, complete, and cancel links
- Create desktop shortcuts (.stx) to quickly launch the Cross-Platform Client
- Set or clear the item as an active process item.

## Creating Change Requests

Development teams create change requests to record problems and enhancement requests and track their resolution or implementation. Many teams have adopted specific guidelines that govern the creation and content of change requests, for example, instructions about what can be entered in the **Component** and **Category** fields. Be sure to follow these guidelines, if they exist.

1. Open a StarTeam perspective, and select the **Change Request** view.
2. Right-click inside the **Change Request** view and choose **New...** The **Change Request** dialog box opens.
3. Define the change request summary information on the **Synopsis** tab.
4. Describe the change request on the **Description** tab.
5. Optionally, specify a temporary workaround on the **Solution** tab.
6. Optionally, specify custom change request properties on the **Custom** tab.
7. Optionally, attach files related to this change request on the **Attachments** tab.
8. Optionally, add comments about the change request on the **Comment** tab.
9. Click **OK**. StarTeam assigns a unique number to the change request and displays the summary information.

## Specifying Change Request Summary Information

You use the **Synopsis** tab to define and modify the summary information about a change request. Summary information includes important criteria like status, severity, and who is currently responsible for this change request.

1. On the **Synopsis** tab, accept the default status **New** or select another status from the **Status** list.
2. Indicate the severity of the change request by selecting **High**, **Medium**, or **Low** from the **Severity** list.



**Note:** The team leader usually sets the criteria for high, medium and low status.

3. If the change request needs immediate attention, select **Yes** from the **Priority** list.
4. To specify the type of change request, select **Defect** or **Suggestion** from the **Type** list.
5. Select the platform to which the change request applies from the **Platform** list.
6. Type a summary of the change request in the **Synopsis** field. The application can accept a maximum of 20K characters in this text box, but your database may accept fewer characters.
7. Select the name of the team member responsible for correcting the change request from the **Responsibility** list.
8. Click **Apply**.

## Specifying Change Request Descriptions

You use the **Description** tab to specify detailed information about the change request including the steps to reproduce the problem.

1. Click the **Description** tab.
2. Type a detailed description of the change request in the **Description and steps to reproduce** field.

Include the steps to reproduce the problem, or in the case of an enhancement request, a detailed description of the enhancement.



3. Optionally, type or browse for the path to a test for the change request in the **Test command** field.
4. Click **Apply**.

## Specifying Change Request Solutions

You use the **Solution** tab to specify a workaround for the problem and to document how this change request was resolved.

1. Click the **Solution** tab.
2. Optionally, in the **Work around** field, type the steps you can follow to work around the problem.
3. Optionally, in the **Fix** field, type the solution to the problem. The **Fix** field is usually completed by the user who fixes the code. In this field, the application can accept a maximum of 20K characters, but your database may accept fewer characters.
4. Click **Apply**.

## Modifying Custom Options for Change Requests

Your team leader may have created additional change request properties. You use the **Custom** tab to change the default properties.

To set the values for custom properties:

1. Double-click a custom property on the **Custom** tab to open the **Edit Property** dialog box.
2. Type or select a new value for the property by double-clicking on the field:

**integer, text, and real fields**      **Value** is a text box.

**enumerated types and user IDs**      **Value** is a list box.

**dates and times**      **Value** has a **Date** check box and a **Time** check box, each of which is followed by a date or time in the format for your locale.


 **Tip:** To enter a blank value for a **GroupList** or **UserList** property, click on a selected row to deselect it. When the item is no longer highlighted, click **OK**.

3. Click **Apply**.


## Adding Change Request Comments

You can add comments to a change request, such as the reason for changing the change request properties.

1. Click the **Comment** tab.

 **Note:** The **Comment** for this revision lists any comments that were entered for the current version of the change request only. That is, each time you change the change request and type a comment, the new comment replaces the old comment when you save the change request.

2. Type your comments in the **Comment for new revision** field.

 **Note:** You must make a change to a property of this change request before you can type a comment.

3. Click **Apply** to save your changes.

## Adding Change Request Attachments

1. Click the **Custom** tab.
2. Click **Add** to attach a file to the change request.
3. Select the files to be attached from the **Open** dialog box.

4. Click **Open**.
5. *Optional:* Add more attachments.
6. Click **Finish**.

## Editing Change Requests

1. Open a StarTeam perspective, and select the **Change Request** view.
2. Double-click the change request. The **Change Request Properties** dialog box opens.
3. *Alternative:* Right-click the change request and choose **Edit**. A **Change Request property** editor opens, embedded in the Eclipse editor. The embedded property editor allows you to open other items and editors at the same time, and enables you to drag and drop files or content from other items into the change request fields. Since the embedded editor is non-modal, you can continue to do other work while it is open.
4. Make any desired changes to the change request properties.
5. Click **Finish** in the **Change Request Properties** dialog box or if using the embedded editor, click **Save** on the main toolbar.



**Note:** To edit the change request, right-click it in the **Change Request** view and choose **Edit**.

## Viewing Unread Change Requests

1. Look for change requests in bold. These are the change requests you are responsible for but have not yet reviewed.



**Note:** Change requests in regular type are those that you have read or those for which you are not responsible.

2. Click on the **Responsibility** column header, then scroll down to your name. All change requests in bold are unread.

## Default and Required Change Request Fields

The following table lists the fields on the **Change Request** dialog box, explains their uses, and indicates which fields are required.

Field	Required?	Description	Example
<b>Status</b>	Yes	For new change requests, set the <b>Status</b> field to <b>New</b> . The <b>Status</b> is changed to <b>Open</b> when the change request is assigned to a developer.	In this example, the status should be <b>New</b> .
<b>Severity</b>	Yes	Specify the seriousness of the problem. High severity items are usually associated with data loss or corruption, system crashes, etc. Low severity items are generally misspelled items and cosmetic errors.	In this example, the problem is comparatively minor (that is, if it does not cause the system to crash or lose data), so classify it as <b>Medium</b> .
<b>Priority</b>	Yes	In most defect tracking systems, <b>Priority</b> is a multi-level choice (usually on a 1 to 5 scale). In StarTeam, however, it is a <b>Yes</b> or <b>No</b> choice. The priority of a change request is sometimes determined by the tester and sometimes by the developer. In most cases, it reflects the need to get a particular defect fixed before others. If the defect is catastrophic or prevents your team from	In this example, leave the <b>Priority</b> field cleared.

Field	Required?	Description	Example
		accessing other major areas of the application, select the <b>Priority</b> field.	
<b>Platform</b>	Yes	Indicate what type of operating system environment the defect occurs in. If the defect happens only on Windows 10, select Windows 10. In most cases, the defect will appear on all platforms.	In this example, set <b>Platform</b> to <b>All</b> .
<b>External Reference</b>	No	Specify information received from outside the company, such as a note about a defect from an outside testing service or a customer.  Currently this field is not used.	In this example, leave the field empty.
<b>Component</b>	No	Identify the component of the product in which the defect occurs.  Currently this field is not used.	In this example, leave the field empty.
<b>Category</b>	No	Identify a sub-component of the product. It is used with the <b>Component</b> field to identify the location in which the defect occurs.  Currently this field is not used.	In this example, leave the field empty.
<b>Synopsis</b>	Yes	Use to give a brief summary of the problem encountered or the suggested enhancement. Consider the synopsis to be a title for the defect.  <b>Note:</b> The <b>Synopsis</b> should only contain information for one defect. If the reported defect uncovers or relates to another defect, the second defect should be written up separately and referenced to the first defect in the synopsis (for example, "CR #3109 also relates to this defect").	For this example, a synopsis might be: "Available fields disappear when using the Advanced Fields box."
<b>Type</b>	Yes	If the change request is a reproducible problem in the software, select <b>Defect</b> . If it is a customer request or a feature enhancement request, select <b>Suggestion</b> .	For this example, select <b>Defect</b> .
<b>Last Build Tested</b>	Yes	Indicate the build number of the software in which the defect was discovered or last tested. If you are writing a change request, select the build number from the application (often found in the <b>About</b> dialog). If you are verifying or regressing the change request, and the problem still exists in the current build, change this field to the build number you are currently testing.	For this example, select the most current build number.
<b>Addressed in Build</b>	Yes	Indicate the build in which the fix first appears. In most cases, after the engineer fixes the defect, the field will be set to <b>Next Build</b> . This field changes to the correct build when that version is actually built.	For this example, leave the field empty.
<b>Responsibility</b>	No	Indicate the person who should act on the defect. Depending on the position of the change request in the change request life cycle, this person could be a developer, a QA engineer, or the person who first reported the change request.	For this example, either leave this field blank, or assign it to the lead engineer on the project, who will assign it to the appropriate person.

Field	Required?	Description	Example
<b>Addressed by</b>	Yes	This field is automatically filled with the name of the person who originally wrote up the change request. It is not editable.	NA
<b>Description/Steps to Reproduce</b>	Yes	<p>Select the <b>Description Tab</b>. In the <b>Description/Steps to Reproduce</b> field, enter detailed information about the defect. Specifically, the description should build on the synopsis information.</p> <p>The <b>Steps to Reproduce</b> information is the most important data entered in the change request because it provides a detailed step-by-step method of reproducing the defect. The more detailed the information, the more likely the responsible developer will be able to determine the cause of the defect and fix the defect.</p>	<p>Steps to reproduce might look as follows:</p> <ol style="list-style-type: none"> <li>1. Click the column headers in the upper pane.</li> <li>2. Select <b>Show Fields</b>.</li> <li>3. Click <b>Show Advanced Fields</b> check box. The check box is activated.</li> <li>4. Click <b>Show Advanced Fields</b> check box. The check box is deactivated.</li> <li>5. EXP: The standard fields appear in the <b>Available Fields</b> list.</li> <li>6. ACT: No fields appear in the <b>Available Fields</b> list.</li> </ol>

## Moving Change Requests

1. Locate the change request you want to move. You can move a change request from one folder to another.
2. Click on the change request and drag it to a new folder.

## Assigning Change Requests

Assigning a change request refers to assigning the status of the change request as well as who is currently responsible for the change request.

1. Select the folder containing the change request in the Server Explorer or in one of the Eclipse Explorers.
2. Select the change request, then choose **Properties** from the **Change Request** menu or context menu. The **Change Request Properties** dialog box opens.
3. Review the settings and decide on an appropriate status. You can select **Open**, **Is Duplicate**, **As Designed**, or **Deferred**.
  - If you select **Open**, the **Responsibility** changes to the person best qualified to fix or enhance the product, as described in the change request.
  - If you select **Is Duplicate** or **As Designed**, the **Responsibility** changes to the person who submitted the change request. The assumption is that the person who submitted the change request will want to know about, verify, or perhaps challenge this change in status.
4. Click **Apply**, then click **Next** or **Previous** to review another change request.

## Resolve Open Change Requests

You resolve open change requests by following the steps below. Before you start working on a change request, be aware of any processes required by your team. For example:

- Your company might require that the change request **Status** be changed to **In Progress**.
  - You might be required to link open change requests to the associated file or files that need to be changed. If this is the case, when you check in a file or group of files, you can indicate the change requests that are being fixed by the files. Doing this saves the time it would take to change the status of each change request.
1. To locate change requests that need to be resolved, select a folder in the Server Explorer or in one of the Eclipse views and then open the **Change Request** view.
  2. Click **Link with Selection**.
  3. Double-click the change request. The **Change Request <number, revision #>** dialog box opens.
  4. Change the **Status** of the change request to one of the resolved statuses: **Fixed**, **Documented**, or **Cannot Reproduce**. Alternatively, you might use **Is Duplicate** or **As Designed**, if either of these is appropriate.

When you select a resolved status, StarTeam automatically makes the following changes to the change request:

- Places the name of the person who submitted the change request in the **Responsibility** field. The assumption is that the person who submitted the change request will want to know about, verify, or perhaps challenge the change in status.
  - Changes the setting for the **Addressed in build** field to **Next Build** (if the status has changed to **Fixed** or **Documented**). When the next build label is created, **Next Build** changes to the name of the build label. The assumption is that the person who verifies that the change request has been implemented should test the correct build of the product.
5. If you choose **Fixed** or **Documented** as the new status, select the **Solution** tab and type the appropriate information in the **Work Around** and/or **Fix** text boxes.

Often a change request suggests one or more fixes for a problem, and none of these suggestions are implemented. To avoid confusion, the fix that is implemented must be described in precise detail. Testers and writers rely heavily on this information.

6. Click **OK**.



**Tip:** Although the application makes these automatic changes immediately, you can change the **Responsibility** or **Addressed in build** setting before you click **OK** (or **Apply**, if appropriate). In this way, you can bypass the automatic workflow and route the change request as your team requires.

## Verifying Resolved Change Requests

You verify resolved change requests by following the steps below. If you determine that a change request is not really resolved, you can reopen it.

1. Open the **Change Request** view.
2. Click **Link with Selection**.
3. Double-click the change request. The **Change Request <number, revision #>** dialog box opens.
4. Change the status to **Open** or **Verified**. StarTeam has the following verified statuses:
  - Verified As Designed
  - Verified Cannot Reproduce
  - Verified Documented
  - Verified Fixed
  - Verified Is Duplicate
5. If you change the status to **Open**, type the word `Reopen` and the date in the **Synopsis** field. Otherwise, the team member who resolved the change request may think that he or she forgot to mark it resolved and, without investigating further, mark it resolved a second time.

When you re-open a change request, StarTeam automatically does the following:

- Places the name of the person who resolved the change request in the **Responsibility** field. The assumption is that the person who resolved the change request the first time should be the person to continue working on it.
  - Blanks out the setting for the **Addressed in build** field. The assumption is that the change request has not been resolved and, therefore, has not been addressed in any build.
6. Do one of the following:
- Click **Apply**, then click the **Next** or **Previous** button to verify another change request.
  - Click **OK**.

## Closing Verified Change Requests

1. Open the **Change Request** view.
2. Click **Link with Selection** in the **Change Request** view toolbar.
3. Select the folder containing the change request in the Server Explorer or in one of the Eclipse Explorers.
4. Double-click the change request. The **Change Request <number, revision #>** dialog box opens.
5. Change the status to **Closed**. StarTeam has the following closed statuses:
  - Closed (As Designed)
  - Closed (Cannot Reproduce)
  - Closed (Deferred)
  - Closed (Documented)
  - Closed (Fixed)
  - Closed (Is Duplicate)
6. Do one of the following:
  - Click **Apply**, then click the **Next** or **Previous** button to close another change request.
  - Click **OK**.

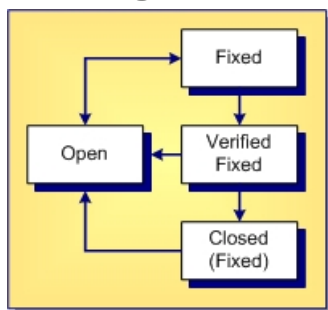
## Reviewing Linked Change Requests

If you are checking in a file that has one or more linked change requests, you should also review all change requests associated with the file.

1. Select the file that is linked to the change request in one of the Eclipse Explorers.
2. Choose **Team > Check In**. The **Check In** dialog box opens.
3. Expand the **Change Requests** section. This section expands and displays a list of any **Change Requests Linked In This View**.
4. Optionally, double-click a change request to review or edit its properties.
5. Optionally, check **Marked Selected Change Requests As Fixed**.

If you check this option, StarTeam will mark the selected, but unresolved, change request **Fixed** as part of the check-in process.

## Change Request Lifecycle



The above diagram shows the lifecycle for a change request with an initial status of *Open*. The status was then set to *Fixed*. After this setting, the built-in workflow added an additional status field of *Verified Fixed*. Finally, the change request was closed, meaning its status was set to *Closed (Fixed)*.

The diagram also shows that a change request can be reopened at any stage in its lifecycle because the arrows leading from each of the three fixed statuses can lead back to the *Open* status at any time.

## Change Request Status Field

StarTeam's built-in workflow for change requests is based on the setting of the **Status** field for the change request. For this reason, you cannot add additional settings to the **Status** field. However, you can rename them to better suit preferences set by your organization. For example, your organization may prefer to change the name of the status *New* to *New Change Request*.

When you alter the status of a change request, the built-in workflow automatically selects the appropriate properties associated with the change in status.

After selecting *New*, *Open*, or *In Progress*, six new statuses display in the Status drop-down list box. These statuses, which are associated with the status you selected, are:

- Deferred
- Cannot Reproduce
- As Designed
- Fixed
- Documented
- Is Duplicate

## Change Request Properties

This topic presents the change request properties and their descriptions as displayed in the **Change Request Properties** dialog box. The **Change Request Properties** dialog box contains the following tabbed pages of properties:

### Synopsis

The following properties are on the **Synopsis** page.

<b>Status</b>	Displays the status of the change request.
<b>Priority</b>	Displays the priority level of the change request. Many people use repository customization to extend this field to include other values because Boolean values in the application are treated as enumerated types. For example, No is 0 and Yes is 1. An administrator might change No to Not A Priority, Yes to Priority 1, and add Priorities 2 through 10.
<b>Type</b>	Displays the type of change request, a <b>Defect</b> or a <b>Suggestion</b> .
<b>Severity</b>	Indicates the severity of the change request: <b>Low</b> , <b>Medium</b> , or <b>High</b> .
<b>Platform</b>	Indicates which operating system platform the to which the change request applies.
<b>Last Build Tested</b>	Displays the build label selected by a user to represent the last build in which a change request was tested.
<b>External Reference</b>	Indicates the customer or other outside source who provided the data for this change request.
<b>Addressed In Build</b>	Indicates the next build label created and applied to the view after the resolution to a change request occurs.

<b>Component</b>	Displays the component in which the defect occurs. It is often used with the <b>Category</b> property to narrow that identification to a sub-component.
<b>Category</b>	Displays the name of the sub-component in which the defect occurs. It is usually used in combination with the <b>Component</b> property.
<b>Synopsis</b>	Displays a brief description of the change request.
<b>Responsibility</b>	Displays the name of the person currently responsible for the change request.
<b>Entered By</b>	Displays the name of the person who entered the change request.

### Description

The following properties are on the **Description** page. This page also contains a **Browse** button for locating the command to test and a **Run** button for running the test.

<b>Description And Steps To Reproduce</b>	Displays a detailed description of the change request.
<b>Test Command</b>	Displays the command to use to test the solution for the change request.

### Solution

The following properties are on the **Solution** page.

<b>Work Around</b>	Explains the solution to the change request other than the fix.
<b>Fix</b>	Displays the solution to the problem addressed by the change request.

### Custom

You can create custom properties for an item which will display in the item **Properties** dialog box.

The following properties are on the **Custom** page.

<b>Property</b>	Displays each custom property name.
<b>Value</b>	Displays the values for each custom property. Double-click the property name to edit the value.

### Attachments

The **Attachments** page contains a list of all the files attached to the current change request.

### Comment

The following properties are on the **Comment** page.

<b>Comment For This Revision</b>	Displays the reason for the changes to the current revision.
<b>Comment For New Revision</b>	Displays the reason for the changes to the new revision.

## Change Request Fields


This section lists all the change request fields in alphabetical order.



**Note:** Client-calculated fields cannot be used in custom email notifications or StarTeam Notification Agent . Reports can use any field name.

<b>Addressed By</b>	Values: list of users, <None>.
---------------------	--------------------------------



	<p>Internal Identifier: <code>AddressedBy</code>.</p> <p>Indicates the user who resolved a change request (resolved statuses are <code>Cannot Reproduce</code>, <code>As Designed</code>, <code>Fixed</code>, <code>Documented</code>, and <code>Is Duplicate</code>).</p>
<b>Addressed In</b>	<p>Values: list of view labels, <code>&lt;None&gt;</code>.</p> <p>Internal Identifier: <code>AddressedIn</code>.</p> <p>Indicates the next build label created and applied to the view after the resolution to a change request occurs.</p>
<b>Addressed In View</b>	<p>Values: list of views, <code>&lt;None&gt;</code>.</p> <p>Internal Identifier: <code>AddressedInView</code></p> <p>Indicates in what view the change request has been resolved. This is important for shared, and perhaps moved, change requests.</p>
<b>Attachment Count</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>AttachmentCount</code>.</p> <p>The number of files attached to an item.</p>
<b>Attachment IDs (Advanced)</b>	<p>Values: <code>byte array</code>. Displayed as a bracketed series of numbers in hex format. For example: <code>[00 00 00 00 02 00 00 00]</code> indicates two specific attachments.</p> <p>Internal Identifier: <code>AttachmentCount</code>.</p> <p>Cannot be used in queries. The ID numbers assigned to attachments. For example, the first attachment within a project is <code>00 00 00 00</code>.</p>
<b>Attachment Names</b>	<p>Values: <code>text</code> containing a series of file names separated by spaces.</p> <p>Internal Identifier: <code>AttachmentNames</code>.</p> <p>The names of the files attached to an item.</p>
<b>Branch On Change (Advanced)</b>	<p>Values: <code>No</code>, <code>Yes</code>.</p> <p>Internal Identifier: <code>BranchOnChange</code>.</p> <p>Indicates whether the item will branch when it changes.</p> <p>The value is <code>No</code> if the item's behavior is not set to <b>Branch On Change</b>. Reasons for this may be:</p> <ul style="list-style-type: none"> <li>• The item is in the root or a reference view and the <b>Branch On Change</b> feature is disabled.</li> <li>• The item is in a branching view but has already branched as a result of a change, which, in turn, results in the <b>Branch On Change</b> feature becoming disabled.</li> <li>• The item is in a branching view, but its behavior currently does not permit it to branch on change. This means that modifications are checked into the parent view.</li> </ul> <p> <b>Note:</b> If the value is <code>No</code>, the value of the <b>Branch State</b> explains the <code>No</code>.</p>
<b>Branch State (Advanced)</b>	<p>Values: <code>Branched</code>, <code>Not Branched</code>, <code>Root</code>.</p> <p>Internal Identifier: <code>BranchState</code>.</p>

Indicates whether an item has branched in the child view, is still unbranched (and therefore is part of the parent view), or was created in the view in which it resides.

The values `Branched` and `Not Branched` apply to items in branching views. The value `Root` applies to items created in the view in which the item currently resides.

If the view is a reference view, it reflects the state of the item in the reference view's parent.

### Category

Values: `text`.

Internal Identifier: `Category`.

Text identifying the sub-component in which the defect occurs. It is usually used in combination with the **Component** field.

### Closed On

Values: `date/atime`.

Internal Identifier: `ClosedOn`.

The date and time at which a change request was closed.

### Comment

Values: `text`.

Internal Identifier: `Comment`.

The initial 2000 characters provided as the reason for changing an item's properties or contents are stored in the **Short Comment** field. The **Comment** field stores those 2000 characters and any additional text. Changing an item's properties causes the application to create a new revision.



**Note:** To include a **Link** comment, the **Comment** field is the value to use in an HTML report.

### CommentID (Advanced)

Values: `number`.

Internal Identifier: `CommentID`.

The ID number assigned to the revision comment. Displays `-1` if no revision comment was supplied.

### Component

Values: `text`.

Internal Identifier: `Component`.

Text identifying the component in which the defect occurs. It is often used with the **Category** field to narrow that identification to a sub-component.

### Configuration Time

Values: `date/time`.

Internal Identifier: `ConfigurationTime`.

Indicates the time to which an item is configured. If you configure an item to a specific time, this field contains that time. If you configure an item to a label or promotion state, this field shows either the time at which the label was created or the time at which the label associated with the promotion state was created.

### CR Number

Values: `number`.

Internal Identifier: `ChangeNumber`.

The number assigned to a change request. For example, if the **Object ID** is 0, the change request number is 1.

<b>Created By</b>	<p>Values: list of users, &lt;None&gt;.</p> <p>Internal Identifier: <code>CreatedUserID</code>.</p> <p>The name of the user who created the first revision in the view. This is either the user who added the item to the project, or the user who checked in the revision that branched.</p>
<b>Created Time</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>CreatedTime</code></p> <p>The time at which the first revision in the view was created.</p>
<b>Deleted By</b>	<p>Values: list of users, &lt;None&gt;.</p> <p>Internal Identifier: <code>DeletedUserID</code>.</p> <p>The name of the user who deleted the item. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.</p>
<b>Deleted Time</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>DeletedTime</code>.</p> <p>The time at which an item was deleted. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.</p>
<b>Description</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>Description</code>.</p> <p>The description provided for an item at the time it was added to the view, including any later edits to it.</p>
<b>Dot Notation</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>DotNotation</code>.</p> <p>The branch revision number, for example, <code>1.2.1.0</code>.</p>
<b>End Modified Time (Advanced)</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>EndModifiedTime</code>.</p> <p>The date and time at which a revision ceased to be the tip revision. Although this field can be displayed in the upper pane, its value is always blank. This is because, at any given configuration time, the item is still the tip revision.</p>
<b>Entered By</b>	<p>Values: list of users, &lt;None&gt;.</p> <p>Internal Identifier: <code>EnteredBy</code>.</p> <p>The name of the user who created this change request.</p>
<b>Entered On</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>EnteredOn</code>.</p> <p>The time at which this change request was created.</p>
<b>External Reference</b>	<p>Values: <code>text</code></p> <p>Internal Identifier: <code>ExternalReference</code>.</p>

	Text usually used to indicate a customer or other outside source who provided the data for this change request.
<b>Fix</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>Fix</code>.</p> <p>The text in the <b>Fix</b> field.</p>
<b>Flag</b>	<p>Values: <code>No</code>, <code>Yes</code>.</p> <p>Internal Identifier: <code>Flag</code>.</p> <p>Marks or bookmarks files in the upper pane on your workstation. This is a client-calculated field.</p>
<b>Flag User List (Advanced)</b>	<p>Values: text displayed as a list of user names. For example: <code>[Greg, Sam]</code> indicates user names.</p> <p>Internal Identifier: <code>FlagUserList</code>.</p> <p>Can be used in queries. Identifies users who have set flags on a given item.</p>
<b>Folder</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>Folder</code>.</p> <p>The name of the folder that stores the item. This is a client-calculated field.</p>
<b>Folder Path</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>Folder Path</code> (contains spaces).</p> <p>The path to the folder. This is not the path to the working folder.</p>
<b>Item Deleted By</b>	<p>Values: list of users, <code>None</code>.</p> <p>Internal Identifier: <code>ItemDeletedUserID</code>.</p> <p>The name of the user who deleted this item. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.</p>
<b>Item Deleted Time</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>ItemDeltedTime</code>.</p> <p>The time at which the item was deleted. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.</p>
<b>Last Build Tested</b>	<p>Values: list of view labels, <code>&lt;None&gt;</code>.</p> <p>Internal Identifier: <code>LastBuildTested</code>.</p> <p>The build label selected by a user to represent the last build in which a change request was tested.</p>
<b>Locked By</b>	<p>Values: list of users, <code>&lt;None&gt;</code>.</p> <p>Internal Identifier: <code>ExclusiveLocker</code>.</p> <p>The name of the user who has exclusively locked a folder.</p>
<b>Modified By</b>	<p>Values: list of users, <code>&lt;None&gt;</code>.</p> <p>Internal Identifier: <code>ModifiedUserID</code>.</p> <p>The name of the user who last modified the item.</p>

<b>Modified Time</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>ModifiedTime</code>.</p> <p>The time at which an item was last modified. The item may have been checked in or had its properties changed. For folders, this has nothing to do with the working folder. Use <b>Local Time Stamp</b> for the time a working folder was last modified.</p>
<b>My Lock</b>	<p>Values: <code>Exclusively Locked By Me, Non-exclusively Locked By Me, Not Locked By Me</code>.</p> <p>Internal Identifier: <code>MyLock</code>.</p> <p>Indicates whether the current user has the item locked and, if so, whether that lock is exclusive or not. This is a client-calculated field.</p>
<b>New Revision Comment (Advanced)</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>NewRevisionComment</code>.</p> <p>Internal use only. The client uses this value during the item update process. The field always appears empty if added to the upper pane. This is a client-calculated field.</p>
<b>Non-Exclusive Lockers</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>NonExclusiveLockers</code>.</p> <p>The names of the users who have locked the folder non-exclusively.</p>
<b>Object ID</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>ID</code>.</p> <p>Each item is assigned an object ID when it is added to a view. For applicable items, when it is branched in a child view, it is assigned another object ID. The original ID belongs to the folder in the parent view.</p>
<b>Parent Branch Revision (Advanced)</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>ParentRevision</code>.</p> <p>The last digit in the branch revision number before an item branched. For example, if this number is 7, the branch revision was 1.7 at the time the item branched (becoming 1.7.1.0, as seen in the item's history). This number is -1 if an item was not inherited from the parent view.</p>
<b>Parent ID (Advanced)</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>ParentID</code>.</p> <p>The object ID of an item in the parent view. The Parent ID is -1 if this view has no parent view.</p>
<b>Parent Revision (Advanced)</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>PathRevision</code>.</p> <p>The revision number at which an item branched. For example, if this number is 8, this item's revision number in the parent view was 8 at the time the item branched. The history should show that revision 9 in the first revision in the current view. This number is 0 if this item was not inherited from the parent view.</p>
<b>Platform</b>	<p>Values: <code>All, MacOS, Other, Unix, Windows 2000, Windows 95, Windows 98, Windows NT, Windows XP</code>.</p>

	<p>Internal Identifier: <code>Platform</code></p> <p>The value of the <b>Platform</b> field.</p>
<b>Priority</b>	<p>Values: <code>No</code>, <code>Yes</code>.</p> <p>Internal Identifier: <code>Priority</code>.</p> <p>The value of the <b>Priority</b> field. Many people use repository customization to extend this field to include other values because booleans in the application are treated as enumerated types. For example, <code>No</code> is 0 and <code>Yes</code> is 1. An administrator might change <code>No</code> to <code>Not A Priority</code>, <code>Yes</code> to <code>Priority 1</code>, and add <code>Priority 2</code> through <code>Priority 10</code>.</p>
<b>Read Only (Advanced)</b>	<p>Values: <code>No</code>, <code>Yes</code>.</p> <p>Internal Identifier: <code>ReadOnly</code>.</p> <p>Indicates whether the item's configuration is read-only (as in a rollback configuration of a view)/its behavior does not allow it to branch on modification. For folders, do not confuse a read-only configuration (an application issue) with a read-only folder (an operating system issue). A read-only folder cannot be edited and saved to disk. A folder whose configuration is read-only can be edited and saved to disk; it just cannot be checked in.</p>
<b>Read Status</b>	<p>Values: <code>Read</code>, <code>Unread</code>.</p> <p>Internal Identifier: <code>ReadStatus</code>.</p> <p>Indicates whether an item is considered read or not read. This is a client-calculated field.</p>
<b>Read Status User List</b>	<p>Values: <code>text</code> displayed as a list of user names. For example: <code>[Greg, Sam]</code> indicates user names.</p> <p>Internal Identifier: <code>ReadStatusUserList</code>.</p> <p>Can be used in queries. Identifies users for whom a given item's status is <b>Unread</b>.</p>
<b>Resolved On</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>ResolvedOn</code>.</p> <p>The time at which a change request was resolved. The resolution can be: <b>Cannot Reproduce</b>, <b>As Designed</b>, <b>Fixed</b>, <b>Documented</b>, or <b>Is Duplicate</b>.</p>
<b>Responsibility</b>	<p>Values: list of users, <code>&lt;None&gt;</code>.</p> <p>Internal Identifier: <code>Responsibility</code>.</p> <p>The name of the user who is currently responsible for a change request.</p>
<b>Revision Flags (Advanced)</b>	<p>Values: 0.</p> <p>Internal Identifier: <code>RevisionFlags</code>.</p> <p>Internal use only.</p>
<b>Root Object ID (Advanced)</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>RootObjectID</code>.</p> <p>The object ID of the oldest ancestor of an item. For example, if an item was not inherited from a parent view, the root object ID is the same as its object ID. If it was</p>

	<p>inherited from a parent view, the root object ID is the Parent ID, or the item's Parent ID.</p>
<b>Severity</b>	<p>Values: High, Low, Medium.</p> <p>Internal Identifier: Severity.</p> <p>The value of the <b>Severity</b> field.</p>
<b>Share State</b>	<p>Values: DerivedShare, Not Shared, Root Share.</p> <p>Internal Identifier: ShareState</p> <p>Indicates whether this item is shared. Not Shared means that the item is not shared. Root Share means that the item is shared and this item is the original (or root) reference. DerivedShare means that the item is shared, but this item is not the original (or root) reference.</p>
<b>Short Comment</b>	<p>Values: text.</p> <p>Internal Identifier: ShortComment.</p> <p>Stores the initial 2000 characters provided as the reason for changing an item's properties or contents. Additional text is stored in the <b>Comment</b> field.</p>
<b>Status</b>	<p>Values: New, Open, In Progress, Deferred, Cannot Reproduce, As Designed, Fixed, Documented, Is Duplicate, Verified Deferred, Verified Cannot Reproduce, Verified As Designed, Verified Fixed, Verified Documented, Verified Is Duplicate, Closed Deferred, Closed Cannot Reproduce, Closed As Designed, Closed Fixed, Closed Documented, Closed Is Duplicate.</p> <p>Internal Identifier: Status.</p> <p>The value of the <b>Status</b> field.</p>
<b>Synopsis</b>	<p>Values: text.</p> <p>Internal Identifier: Synopsis.</p> <p>The value of the <b>Synopsis</b> field.</p>
<b>Test Command</b>	<p>Values: text.</p> <p>Internal Identifier: TestCommand.</p> <p>The text in the <b>Test Command</b> field.</p>
<b>Type</b>	<p>Values: Defect, Suggestion.</p> <p>Internal Identifier: Type.</p> <p>The value of the <b>Type</b> field.</p>
<b>Verified On</b>	<p>Values: date/time.</p> <p>Internal Identifier: VerifiedOn.</p> <p>The time at which a change request was verified. The resolution can be <b>Verified Cannot Reproduce, Verified As Designed, Verified Fixed, Verified Documented, or Verified Is Duplicate.</b></p>
<b>Version (Advanced)</b>	<p>Values: number.</p> <p>Internal Identifier: RevisionNumber.</p>

The last number in the branch revision number. For example, if the branch revision number is 1.3.1.2, the version is 2.

#### View

Values: list of views, <None>.

Internal Identifier: ViewID.

The name of the view in which the item last branched. For example, if an item is inherited from a parent view but is branched in a child view, the value of this field in the child view changes from the name of the parent view to the name of the child view for the revision that branched and subsequent revisions in the child view.

#### Work Around

Values: text.

Internal Identifier: WorkAround.

The text in the **Work Around** field.

## Commonly Used Change Request Abbreviations

When using a defect tracking system, most organizations employ a special syntax or shorthand to describe the steps required to reproduce a problem.

Shorthand	Description
1, 2, 3,...	Number the steps.
LClick	Click the left mouse button.
RClick	Click the right mouse button.
DClick	Double click the left mouse button.
[ ]	A keyboard button to be pressed. For example, [ F1 ] for Help or [ F5 ] for Refresh.
< >	A dialog button. For example: Press <OK> or <Cancel>.
>	Menu separator. For example: Select File > Open or Topic > Tools > Reports.
//	A comment that is not an actual step. For example: 3) LClick the field. // At this point my machine started to smoke.
EXP	Expected results. For example: EXP: The focus moves to the next field.
ACT	Actual results. For example: ACT: The application crashes.

## Sample Change Request Detail View Template

Use the following sample for customizing the **Detail** view for the change request component.



Cut and paste the HTML contents below into a file named `changerequest.details.html`, and place it in the `Application Data\Borland\StarTeam` folder for the current user. For example, on a Microsoft Windows system, this file would be located in the `C:\Documents and Settings\USER\Application Data\Micro Focus\StarTeam` folder.

### Sample Change Request HTML Template

```

<html>
<head></head>
<body>
<table width=100% border=1>
<tr bgcolor=#aabbcc>
<th>CR Number</th>
<th>Status</th>
<th>Priority</th>
<th>Type</th>
<th>Responsibility</th>
</tr><tr>
<td align=center>~~ChangeNumber~~</td>
<td align=center>~~Status~~</td>
<td align=center>~~Priority~~</td>
<td align=center>~~Type~~</td>
<td align=center>~~Responsibility~~</td>
</tr>
</table>
<p align=right>
<b>Entered By</b>: ~~EnteredBy~~, ~~EnteredOn~~</p>
<b>Synopsis</b>: <br> ~~Synopsis~~<br><br>
<b>Description</b>: <br> ~~Description~~<br><br>
<b>Work Around</b>: <br> ~~WorkAround~~<br><br>
<b>Fix</b>: ~~Fix~~<br><br><hr>
<i>Last modified by: ~~ModifiedUserID~~,~~ModifiedTime~~</i><br>
<b>Number of attachments</b>: ~~AttachmentCount~~<br>
<!--
<b>Flag User List</b>: ~~FlagUserList~~<br>
<b>Version</b>: ~~RevisionNumber~~<br>
<b>Branch State</b>: ~~BranchState~~<br>
<b>Read Status User List</b>: ~~ReadStatusUserList~~<br>
<b>Branch On Change</b>: ~~BranchOnChange~~<br>
<b>Attachment IDs</b>: ~~AttachmentIDs~~<br>
<b>Closed On</b>: ~~ClosedOn~~<br>
<b>Component</b>: ~~Component~~<br>
<b>Parent ID</b>: ~~ParentObjectID~~<br>
<b>Root Object ID</b>: ~~RootObjectID~~<br>
<b>Created Time</b>: ~~CreatedTime~~<br>
<b>Share State</b>: ~~ShareState~~<br>
<b>CommentID</b>: ~~CommentID~~<br>
<b>Folder</b>: ~~Folder~~<br>
<b>Created By</b>: ~~CreatedUserID~~<br>
<b>Deleted Time</b>: ~~DeletedTime~~<br>
<b>Dot Notation ID</b>: ~~DotNotationID~~<br>
<b>Parent Revision</b>: ~~PathRevision~~<br>
<b>Last Build Tested</b>: ~~LastBuildTested~~<br>
<b>Non-Exclusive Lockers</b>: ~~NonExclusiveLockers~~<br>
<b>Short Comment</b>: ~~ShortComment~~<br>
<b>Locked By</b>: ~~ExclusiveLocker~~<br>
<b>Folder Path</b>: ~~Folder Path~~<br>
<b>Object ID</b>: ~~ID~~<br>
<b>Flag</b>: ~~Flag~~<br>
<b>Platform</b>: ~~Platform~~<br>
<b>Severity</b>: ~~Severity~~<br>
<b>Read Only</b>: ~~ReadOnly~~<br>
<b>My Lock</b>: ~~MyLock~~<br>
<b>Configuration Time</b>: ~~ConfigurationTime~~<br>

```

```

<b>Comment</b>: ~~Comment~~<br>
<b>Revision Flags</b>: ~~RevisionFlags~~<br>
<b>Parent Branch Revision</b>: ~~ParentRevision~~<br>
<b>External Reference</b>: ~~ExternalReference~~<br>
<b>Category</b>: ~~Category~~<br>
<b>End Modified Time</b>: ~~EndModifiedTime~~<br>
<b>New Revision Comment</b>: ~~NewRevisionComment~~<br>
<b>Addressed In</b>: ~~AddressedIn~~<br>
<b>Resolved On</b>:~~ResolvedOn~~<br>
<b>View</b>: ~~ViewID~~<br>
<b>Addressed In View</b>: ~~AddressedInView~~<br>
<b>Addressed By</b>: ~~AddressedBy~~<br>
<b>Verified On</b>: ~~VerifiedOn~~<br>
<b>Deleted By</b>: ~~DeletedUserID~~<br>
<b>Test Command</b>: ~~TestCommand~~<br>
<b>Attachment names</b>: ~~AttachmentNames~~<br>
<b>Dot Notation</b>: ~~DotNotation~~<br>
<b>Read Status</b>: ~~ReadStatus~~<br>
-->
</body>
</html>

```

### Fields Used in Detail View Templates

The fields used in the Detail view HTML templates are recognized by the client when they are contained between double tilde ~~ characters. For example: ~~Status~~ represents the Status field found in the **Change Request Properties** dialog box. Refer to the link at the bottom of this topic for more information about the fields that you can use in the Detail view templates.

## Requirements

Requirements are supported for the Enterprise Advantage license and display in the **Requirement** tab of the upper pane in the clients. With the requirement component, you can create requirements within the application and show the dependencies among them. For example, if one requirement must be fulfilled before a second requirement can be fulfilled, the first can be made a child of the second. If your company enforces process rules, the requirements you establish can also be used to drive the development process. Administrators and other authorized users can publish requirements from Caliber to StarTeam using Publisher to StarTeam, which is delivered with Caliber.

### Requirement Characteristics

The requirements in the upper pane have the following characteristics:

- They are attached to the folder selected from the folder hierarchy.
- They match the filter selected from the **Filter** list.
- They match the depth specified by **All Descendants**.



**Note:** You can click the **All descendants** button on the **Requirements** view toolbar.



**Note:** Icons display to the left of a requirement in the upper pane to indicate its status and whether you have read the latest revision.

### How Requirements Can Help

By using a requirements-driven development processes, companies can prevent consuming, costly misunderstandings and shorten time to market. To accomplish this, you can use the StarTeam built-in requirement component as your basic tool, or publish complex requirements to StarTeam from Caliber. Using requirements enables business analysts, managers, developers, QA staff, and others to:

- Organize business, user, and functional requirements in a hierarchical format.
- Indicate the dependencies among requirements.
- See all layers of requirements at all times.
- Prioritize requirements by importance.
- Identify the impact of changes to requirements.
- Use requirements to estimate work.
- Identify the person creating the requirement.
- Notify those who will be responsible for fulfilling the requirements.
- Track the requirement life-cycle from submitted to completed or rejected.
- Provide requirements with a context by linking them to files, change requests, and topics.

## Requirement View

You can open the **Requirement** view by doing one of the following:

- Select **Window > Show View > Requirement**
- Select **Window > Show View > Other**, and choose **Requirement** from the StarTeam tree node.

When you open the **Requirement** view, a list of requirements displays.

All requirements shown in the **Requirement** view have the following characteristics:

- Are attached to the folder selected from the Server Explorer.
- Match the filter selected from the **Filter** list box.
- Match the depth specified by the **All Descendants** button.

### Requirement View Toolbar and Context Menu Actions

The Requirement view toolbar and context menu allow you to accomplish the following tasks:

- Apply predefined filters for viewing lists of requirements
- Search (CTRL+F) for requirements
- Link (unlink) the view to the selected item
- Optionally show all descendants to determine the depth for which the client displays information
- View in list or tree format
- Select all items
- Select items using a label or pre-defined query
- Determine which fields to display in the view
- Perform up to fourth order sorts
- Create, edit, copy, or delete queries
- Save or reset current view filters
- Create, edit, save, or delete filters
- Create new requirements and child requirements
- Open requirements in various views, including History, Label, Link, and Reference views
- Modify and review requirement properties
- Email a text representation of the requirement's properties, along with additional text. The information sent includes the fields displayed in the view.
- Compare the properties of two requirements
- Set the lock status
- Mark the item or thread as read or unread
- Add, edit, enable, or disable custom fields
- Create, attach, and detach labels
- Flag and remove flags from files
- Delete requirements

- Create, complete, and cancel links
- Set or clear the item as an active process item

## Creating Requirements

Creating a hierarchy of requirements allows you to organize a project efficiently and work toward agreed-upon goals.

1. Open the **Requirement** view and click **Link with Selection**.
2. Select the folder in the Server Explorer or in one of the Eclipse Explorers where you wish to create the requirement.
3. Right-click inside the **Requirement** view and select **New...** or **New in Editor** to create a new requirement that is not the child of an existing.
4. Click the **Requirement** tab of the **New Requirement** dialog box and do the following:
  - a) Type a name for the requirement.
  - b) Select an owner (for example, the person ultimately responsible for the fulfillment of the requirement) from the **Owner** list.
  - c) Optionally, provide an external source or reference for the requirement in the **External reference** field. If you publish requirements from CaliberRM to StarTeam, this field displays the CaliberRM identification for this requirement.
  - d) Type the initial description of this requirement in the **Description** field. This description is usually revised over time to eliminate ambiguities.
5. Click the **Responsibility** tab and list the team members responsible for this requirement. If notification is enabled, these people will be notified about changes made to any field in the requirement.
  - a) Click **Add** to display the **Select Responsible Users** dialog box.
  - b) Double-click the name of each person to be added to the list. When you double-click the name, it moves from the **Users** list to the **Responsible Users** list.
  - c) Add the remaining responsible users to the **Responsible Users** list box and click **OK**.
6. Use the **Estimate** tab to indicate the best-case and worst-case times for fulfilling this requirement. The entries are usually in staff days.
  - Type the number of units (usually days) estimated for the fulfillment of this requirement in the **Expected effort** text box.
  - Type the number of units (usually days) estimated for the worst-case fulfillment of this requirement in the **High effort** text box.
  - Type the number of units (usually days) estimated for the best-case fulfillment of this requirement in the **Low effort** text box.
  - Add any appropriate notes in the **Notes** text box.
7. Use the **Custom** tab to provide values for any custom requirement properties that your team leader or company may have created. Double-click a custom property on the **Custom** tab to open the **Edit Property** dialog box.

**integer, text, and real fields**      **Value** is a text box.

**enumerated types and user IDs**      **Value** is a list box.

**dates and times**      **Value** has a **Date** check box and a **Time** check box, each of which is followed by a date or time in the format for your locale.

 **Tip:** To enter a blank value for a **GroupList** or **UserList** property, click on a selected row to deselect it. When the item is no longer highlighted, click **OK**.

Click **Apply** to save your changes.

8. Use the **Comment** tab to explain why the requirement is being created or revised. Enter your reasons in the **Comment for new revision** text box.

## 9. Click OK.

# Sample Requirement Detail View Template

You can use the following sample for customizing the **Detail** view for the requirement component.

```
<html>
<head></head>
<body>
<b>Requirement</b>: #~~Number~~<br>
<b>Name</b>: ~~Name~~<br>
<b>Version</b>: ~~RevisionNumber~~<br>
<b>Modified By</b>: ~~ModifiedUserID~~<br>
<b>Modified On</b>: ~~ModifiedTime~~<br>
<b>Comment</b>: ~~Comment~~<br>
<b>Created By</b>: ~~CreatedUserID~~<br>
<b>Created On</b>: ~~CreatedTime~~<br>
<b>Owner</b>: ~~Owner~~<br>
<b>Status</b>: ~~Status~~<br>
<b>Priority</b>: ~~Priority~~<br>
<b>Locked By</b>: ~~ExclusiveLocker~~<br>
<b>Description</b>: ~~Description~~<br>
<b>Attachments</b>: ~~AttachmentLinks~~<br>
<b>Attachment Count</b>: ~~AttachmentCount~~<br>
<b>Attachment names</b>: ~~AttachmentNames~~<br>
<b>Read Status User List</b>: ~~ReadStatusUserList~~<br>
<b>Share State</b>: ~~ShareState~~<br>
<b>CommentID</b>: ~~CommentID~~<br>
<b>Disabled</b>: ~~Disabled~~<br>
<b>Children Count</b>: ~~ChildrenCount~~<br>
<b>Child Type</b>: ~~ChildType~~<br>
<b>Non-Exclusive Lockers</b>: ~~NonExclusiveLockers~~<br>
<b>Short Comment</b>: ~~ShortComment~~<br>
<b>Recipient Count</b>: ~~RecipientCount~~<br>
<b>Folder Path</b>: ~~Folder Path~~<br>
<b>Object ID</b>: ~~ID~~<br>
<b>Flag</b>: ~~Flag~~<br>
<b>Read Only</b>: ~~ReadOnly~~<br>
<b>My Lock</b>: ~~MyLock~~<br>
<b>Configuration Time</b>: ~~ConfigurationTime~~<br>
<b>End Modified Time</b>: ~~EndModifiedTime~~<br>
<b>Am I Responsible?</b>: ~~AmIResponsible~~<br>
<b>New Revision Comment</b>: ~~NewRevisionComment~~<br>
<b>Type</b>: ~~Type~~<br>
<b>Attachment IDs</b>: ~~AttachmentIDs~~<br>
<b>Dot Notation</b>: ~~DotNotation~~<br>
<b>Read Status</b>: ~~ReadStatus~~<br>
<b>Parent Requirement ID</b>: ~~ParentRequirementID~~<br>
<b>Ambiguities Found</b>: ~~AmbiguitiesFound~~<br>
<b>Expected Effort</b>: ~~ExpectedEffort~~<br>
<b>External Reference</b>: ~~ExternalReference~~<br>
<b>High Effort</b>: ~~HighEffort~~<br>
<b>Low Effort</b>: ~~LowEffort~~<br>
<b>Notes</b>: ~~Notes~~<br>
<b>Responsible Count</b>: ~~ResponsibleCount~~<br>
<b>Responsible Names</b>: ~~ResponsibleNames~~<br>
<b>Revised Description</b>: ~~RevisedDescription~~<br>
</body>
</html>
```

## Fields Used in Detail View Templates

The fields used in the **Detail** view HTML templates are recognized by the client when they are contained between double tilde ~~ characters. For example: ~~Status~~ represents the **Status** field found in the

Requirement dialog box. Refer to the link at the bottom of this topic for more information about the fields that you can use in the Detail view templates.

## Requirement Properties

This topic presents the requirement properties and their descriptions as displayed in the **Requirement Properties** dialog box. The **Requirement Properties** dialog box contains the following tabbed pages of properties.

### Requirement

The following properties are on the **Requirement** page.

<b>Name</b>	Displays the requirement name.
<b>Created By</b>	Displays the name of person who created the first revision of the requirement in the view.
<b>Created On</b>	Displays the date on which first revision of the requirement was created.
<b>Attachments</b>	Indicates the number of files attached to the requirement.
<b>Modified By</b>	Displays the name of the last person who last modified the requirement.
<b>Modified On</b>	Displays the date on which the requirement was last modified.
<b>Type</b>	Displays the requirement type.
<b>Owner</b>	Displays the name of person ultimately responsible for the fulfillment of the requirement.
<b>Status</b>	Displays the current status of the requirement. This indicates the progress from submitted to rejected or completed. Note: The status <b>ReadyForCCP</b> means the requirement is ready for review by the Change Control Board.
<b>External Reference</b>	External source or reference for this requirement. This usually is the name of an external customer who asked for the requirement. If you are publishing requirements from CaliberRM to StarTeam, this property displays its identification for this requirement.
<b>Description</b>	Provides a description of the requirement, usually revised over time to eliminate ambiguities.

### Responsibility

The **Responsibility** page lists the people responsible for completion of the requirement. You can add or remove people from the list.

These people will be notified of changes to the requirement if notification is enabled.

### Ambiguity Review

The following properties are on the **Ambiguity Review** page. Reviewers will use the **Ambiguity Review** page to locate ambiguities in the initial description and revise that description.

<b>Number Of Ambiguities Found</b>	Indicates the number of ambiguities reviewers have found in the initial description of the requirement.
<b>Revised Description</b>	Provides a new, revised description because of ambiguities found in the original description or for other reasons.
<b>Comments</b>	Provides comments stating what the ambiguities are in the original requirement and why you have made the changes to the description.

## Estimate

The following properties are on the **Estimate** page.

- Expected Effort** Indicates the expected case estimate for how long it will take to implement the requirement fully. If you are publishing requirements from CaliberRM to StarTeam, these fields will already be filled with data based on a specific unit, such as hours or days. Otherwise, the units are arbitrary, but should be the same for the **Low Effort** and the **High Effort** fields, and should be used consistently for all requirements.
- High Effort** Indicates the worst case estimate for how long it will take to implement the requirement fully. If you are publishing requirements from CaliberRM to StarTeam, these fields will already be filled with data based on a specific unit, such as hours or days. Otherwise, the units are arbitrary, but should be the same for the **Low Effort** and the **Expected Effort** properties, and should be used consistently for all requirements.
- Low Effort** Indicates the best case estimate for how long it will take to implement the requirement fully. If you are importing requirements from CaliberRM, these fields will already be filled with data based on a specific unit, such as hours or days. Otherwise, the units are arbitrary, but should be the same for the **Expected Effort** and the **High Effort** fields, and should be used consistently for all requirements.

## Custom

You can create custom properties for an item which will display in the item **Properties** dialog box.

The following properties are on the **Custom** page.

- Property** Displays each custom property name.
- Value** Displays the values for each custom property. Double-click the property name to edit the value.

## Attachments

The **Attachments** page contains a list of all the files attached to the current requirement.

## Comment

The following properties are on the **Comment** page.

- Comment For This Revision** Displays the reason for the changes to the current revision.
- Comment For New Revision** Displays the reason for the changes to the new revision.


# Requirement Fields

This section lists all the requirement fields in alphabetical order.



**Note:** Client-calculated fields cannot be used in custom email notifications or StarTeam Notification Agent . Reports can use any field name.

- Am I Responsible?** Values: No, Yes.  
Internal Identifier: AmIResponsible.  
Indicates whether the logged-on user is responsible for a requirement. This is a client-calculated field.
- Ambiguities Found** Values: number.  
Internal Identifier: AmbiguitiesFound.

	Indicates the number of ambiguities found in the requirement.
<b>Attachment Count</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>AttachmentCount</code>.</p> <p>The number of files attached to an item.</p>
<b>Attachment IDs (Advanced)</b>	<p>Values: <code>byte array</code>. Displayed as a bracketed series of numbers in hex format. For example: <code>[00 00 00 00 02 00 00 00]</code> indicates two specific attachments.</p> <p>Internal Identifier: <code>AttachmentCount</code>.</p> <p>Cannot be used in queries. The ID numbers assigned to attachments. For example, the first attachment within a project is <code>00 00 00 00</code>.</p>
<b>Attachment Names</b>	<p>Values: <code>text</code> containing a series of file names separated by spaces.</p> <p>Internal Identifier: <code>AttachmentNames</code>.</p> <p>The names of the files attached to an item.</p>
<b>Children Count</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>ChildrenCount</code>.</p> <p>The number of items that are children of this item. This is a client-calculated field.</p>
<b>ChildType</b>	<p>Values: <code>Child Requirement, Requirement</code>.</p> <p>Internal Identifier: <code>ChildType</code>.</p> <p>Indicates whether the requirement is the root of a requirement tree or a child of another requirement. This is a client-calculated field.</p>
<b>Comment</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>Comment</code>.</p> <p>The initial 2000 characters provided as the reason for changing an item's properties or contents are stored in the <b>Short Comment</b> field. The <b>Comment</b> field stores those 2000 characters and any additional text. Changing an item's properties causes the application to create a new revision.</p> <p> <b>Note:</b> To include a <b>Link</b> comment, the <b>Comment</b> field is the value to use in an HTML report.</p>
<b>CommentID (Advanced)</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>CommentID</code>.</p> <p>The ID number assigned to the revision comment. Displays <code>-1</code> if no revision comment was supplied.</p>
<b>Comments</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>Comments</code></p> <p>Provides comments about the revised description created because of ambiguities found in the original description or for other reasons.</p>
<b>Configuration Time</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>ConfigurationTime</code>.</p>



Indicates the time to which an item is configured. If you configure an item to a specific time, this field contains that time. If you configure an item to a label or promotion state, this field shows either the time at which the label was created or the time at which the label associated with the promotion state was created.

**Created By**

Values: list of users, <None>.

Internal Identifier: `CreatedUserID`.

The name of the user who created the first revision in the view. This is either the user who added the item to the project, or the user who checked in the revision that branched.

**Created Time**

Values: `date/time`.

Internal Identifier: `CreatedTime`

The time at which the first revision in the view was created.

**Deleted By**

Values: list of users, <None>.

Internal Identifier: `DeletedUserID`.

The name of the user who deleted the item. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.

**Deleted Time**

Values: `date/time`.

Internal Identifier: `DeletedTime`.

The time at which an item was deleted. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.

**Description**

Values: `text`.

Internal Identifier: `Description`.

The description provided for an item at the time it was added to the view, including any later edits to it.

**Disabled**

Values: `No`, `Yes`.

Internal Identifier: `Disabled`.

Indicates whether the requirement is disabled.

**Dot Notation**

Values: `text`.

Internal Identifier: `DotNotation`.

The branch revision number, for example, `1.2.1.0`.

**End Modified Time (Advanced)**

Values: `date/time`.

Internal Identifier: `EndModifiedTime`.

The date and time at which a revision ceased to be the tip revision. Although this field can be displayed in the upper pane, its value is always blank. This is because, at any given configuration time, the item is still the tip revision.

**Expected Effort**

Values: `number`.

Internal Identifier: `ExpectedEffort`.

Indicates the expected case estimate for how long it will take to implement the requirement fully. If you are publishing requirements from CaliberRM to StarTeam, these fields will already be filled with data based on a specific unit, such as hours or days. Otherwise, the units are arbitrary, but should be the same for the **Low Effort** and the **High Effort** fields, and should be used consistently for all requirements.

**External Reference**

Values: `text`.

Internal Identifier: `ExternalReference`.

Usually provides the name of an external customer who asked for this requirement.

**Flag**

Values: `No`, `Yes`.

Internal Identifier: `Flag`.

Marks or bookmarks files in the upper pane on your workstation. This is a client-calculated field.

**Flag User List (Advanced)**

Values: text displayed as a list of user names. For example: `[Greg, Sam]` indicates user names.

Internal Identifier: `FlagUserList`.

Can be used in queries. Identifies users who have set flags on a given item.

**Folder Path**

Values: `text`.

Internal Identifier: `Folder Path` (contains spaces).

The path to the folder. This is not the path to the working folder.

**High Effort**

Values: `number`.

Internal Identifier: `HighEffort`.

Indicates the worst case estimate for how long it will take to implement the requirement fully. If you are publishing requirements from CaliberRM to StarTeam, these fields will already be filled with data based on a specific unit, such as hours or days. Otherwise, the units are arbitrary, but should be the same for the **Low Effort** and the **Expected Effort** fields, and should be used consistently for all requirements.

**Item Deleted By**

Values: list of users, `None`.

Internal Identifier: `ItemDeletedUserID`.

The name of the user who deleted this item. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.

**Item Deleted Time**

Values: `date/time`.

Internal Identifier: `ItemDeletedTime`.

The time at which the item was deleted. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.

**Locked By**

Values: list of users, `<None>`.

Internal Identifier: `ExclusiveLocker`.

The name of the user who has exclusively locked a folder.

**Low Effort**

Values: `number`.

Internal Identifier: `LowEffort`.

Indicates the best case estimate for how long it will take to implement the requirement fully. If you are publishing requirements from CaliberRM to StarTeam, these fields will already be filled with data based on a specific unit, such as hours or days. Otherwise, the units are arbitrary, but should be the same for the **Expected Effort** and the **High Effort** fields, and should be used consistently for all requirements.

**Modified By**

Values: list of users, <None>.

Internal Identifier: ModifiedUserID.

The name of the user who last modified the item.

**Modified Time**

Values: date/time.

Internal Identifier: ModifiedTime.

The time at which an item was last modified. The item may have been checked in or had its properties changed. For folders, this has nothing to do with the working folder. Use **Local Time Stamp** for the time a working folder was last modified.

**My Lock**

Values: Exclusively Locked By Me, Non-exclusively Locked By Me, Not Locked By Me.

Internal Identifier: MyLock.

Indicates whether the current user has the item locked and, if so, whether that lock is exclusive or not. This is a client-calculated field.

**Name**

Values: text.

Internal Identifier: Name.

Displays the name of the item.

**New Revision Comment (Advanced)**

Values: text.

Internal Identifier: NewRevisionComment.

Internal use only. The client uses this value during the item update process. The field always appears empty if added to the upper pane. This is a client-calculated field.

**Non-Exclusive Lockers**

Values: text.

Internal Identifier: NonExclusiveLockers.

The names of the users who have locked the folder non-exclusively.

**Notes**

Values: text.

Internal Identifier: Notes.

Text comments on the effort levels for this item.

**Number**

Values: number.

Internal Identifier: RequirementNumber.

Number identifying the requirement. For example, if the Object ID is 0, the requirement number is 1.

**Object ID**

Values: number.

Internal Identifier: ID.

Each item is assigned an object ID when it is added to a view. For applicable items, when it is branched in a child view, it is assigned another object ID. The original ID belongs to the folder in the parent view.

**Owner**

Values: list of users, <None>.

Internal Identifier: `Owner`.

Indicates who is ultimately responsible for this requirement.

**Parent ID  
(Advanced)**

Values: number.

Internal Identifier: `ParentID`.

The object ID of an item in the parent view. The Parent ID is -1 if this view has no parent view.

**Priority**

Values: `Desirable`, `Essential`, `Unassigned`, `Useful`.

Internal Identifier: `Priority`.

The value of the **Priority** field. You can use repository customization to change the names of these values or include other values.

**Read Only  
(Advanced)**

Values: `No`, `Yes`.

Internal Identifier: `ReadOnly`.

Indicates whether the item's configuration is read-only (as in a rollback configuration of a view)/its behavior does not allow it to branch on modification. For folders, do not confuse a read-only configuration (an application issue) with a read-only folder (an operating system issue). A read-only folder cannot be edited and saved to disk. A folder whose configuration is read-only can be edited and saved to disk; it just cannot be checked in.

**Read Status**

Values: `Read`, `Unread`.

Internal Identifier: `ReadStatus`.

Indicates whether an item is considered read or not read. This is a client-calculated field.

**Read Status User  
List**

Values: text displayed as a list of user names. For example: `[Greg, Sam]` indicates user names.

Internal Identifier: `ReadStatusUserList`.

Can be used in queries. Identifies users for whom a given item's status is **Unread**.

**Responsible  
Count**

Values: number.

Internal Identifier: `ResponsibleCount`.

The number of users who are responsible for a requirement.

**Responsible IDs**

Values: byte array, displayed as a bracketed series of numbers in hex format. For example, `[14 00 00 00]` indicates a specific user.

Internal Identifier: `ResponsibleIDs`.

Can not be used in queries. The ID numbers assigned to the users who are responsible for the requirement.

**Responsible  
Names**

Values: text containing a series of user names separated by spaces.

	<p>Internal Identifier: <code>ResponsibleNames</code>.</p> <p>The names of the users responsible for this requirement.</p>
<b>Reviewed By</b>	<p>Values: <code>byte array</code>.</p> <p>Internal Identifier: <code>ReviewedByIDs</code>.</p> <p>Can not be used in queries. Should not be used at all.</p>
<b>Revised Description</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>RevisedDescription</code>.</p> <p>Provides a new, revised description because of ambiguities found in the original description or for other reasons.</p>
<b>Revision Flags (Advanced)</b>	<p>Values: <code>0</code>.</p> <p>Internal Identifier: <code>RevisionFlags</code>.</p> <p>Internal use only.</p>
<b>Share State</b>	<p>Values: <code>DerivedShare, Not Shared, Root Share</code>.</p> <p>Internal Identifier: <code>ShareState</code></p> <p>Indicates whether this item is shared. <code>Not Shared</code> means that the item is not shared. <code>Root Share</code> means that the item is shared and this item is the original (or root) reference. <code>DerivedShare</code> means that the item is shared, but this item is not the original (or root) reference.</p>
<b>Short Comment</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>ShortComment</code>.</p> <p>Stores the initial 2000 characters provided as the reason for changing an item's properties or contents. Additional text is stored in the <b>Comment</b> field.</p>
<b>Status</b>	<p>Values: <code>Accepted, Approved, Complete, Deferred, Draft, Pending, ReadyForCCB, Rejected, Review, Submitted</code>.</p> <p>Internal Identifier: <code>Status</code>.</p> <p>Indicates the status of this requirement.</p>
<b>Type</b>	<p>Values: <code>Business Requirement, Business Specification, Hardware Requirement, Hardware Specification, Human Resources, Information Technology, Software Requirement, Software Specification</code>.</p> <p>Internal Identifier: <code>Type</code>.</p> <p>Indicates the type of requirement.</p>
<b>Version (Advanced)</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>RevisionNumber</code>.</p> <p>The last number in the branch revision number. For example, if the branch revision number is <code>1.3.1.2</code>, the version is <code>2</code>.</p>

# Tasks

The *task* component allows the creation of task lists and work assignments. As a standalone, the task component is very useful for managing a project. It allows team members to indicate who should do what and when, see current task status, estimate hours required to complete a task, record hours spent completing the task, and compare estimated to actual times. Because the application contains both a version control system and a change request system, it also allows tasks to be linked to the files and product defects or suggestions with which they are associated.

The task component can be used independently or interoperate with data from Microsoft Project. It can display tasks in a tree format, which clearly shows the relationship between tasks and subtasks, or in a list format, which allows tasks to be sorted, grouped, or queried, or specific fields to be selected for display. To improve efficiency, each task displays icons that identify its status, priority, milestone, and need for attention. For information about interoperating with Microsoft Project, see the *StarTeam Microsoft Project Integration User's Guide*.

With the StarTeam task component, you can create an individual task or a summary task that has a set of subtasks. It is recommended that you plan tasks before entering them because:

- A task that has even one subtask cannot have work records added to it, although work records can be added to subtasks. The application assumes that the name of the task indicates a goal, perhaps a milestone, that will be reached when the subtasks are completed.
- After a work record has been added to a task, you cannot create subtasks for it.



**Note:** Regardless of whether work records can be added to a task, you should assign the responsibility for its completion to a specific team member. If work records can be added to a task, you should also estimate how long the task should take.

## Creating Tasks

1. Open the **Task** view and click **Link with Selection**.
2. Click **Task > New**. The **New Task** dialog box opens.
3. On the **Task** page of the **New Task** dialog box, type the **Name** of the task. You can use up to 255 characters for the task name.



**Note:** Although the **Responsibility** field names the person primarily responsible for the completion of a task, additional people can be designated on the **Resources** tab.

4. Optionally, check **Milestone** to indicate that the task should be treated as a milestone task. You can display the **Milestone** column in the task list and sort for the tasks that have been designated as milestones.
5. Select the current status of the task from the **Status** list.

**Pending**            Waiting for completion of a predecessor task.

**Ready To Start**    Work can be started on the task.

**In Progress**        Work has been entered for the task.

**Finished**            Work is finished on the task.

**Closed**             Task is completed and closed.

**Hold**                Work temporarily stopped on the task, usually to wait for completion of another task.

6. Choose a **Priority** level from the list. The priorities are identical to those used in Microsoft Project.



**Note:** Do not use the priority **Do Not Level**. This priority is a Microsoft Project-specific term.

7. Type the **Duration** which is the number of hours the task will take to complete. This field is disabled if the task contains sub-tasks since the duration of a task is dependent upon the duration of its sub-tasks.
8. Type the **Percent Complete** which is the percentage of work already completed for this task. This field may range from 0-100. The default is 0 for new tasks.
9. Optionally, check **Needs Attention** to notify team leaders or task reviewers that this task requires attention. Explain why this task needs attention in the text box. Team leaders can add the **Needs Attention** column to their task list and sort for items with this designation.
10. Click **OK**. This creates a new task which will serve as the root of a task tree in the **Task** pane.

## Assigning Task Resources

As you create a task or sub-task, you can assign additional team members as resources to assist in the completion of the task. Use the **Resources** tab to review the list of team members available for this task assignment.

1. Open the **Task** view and click **Link with Selection**.
2. Select the folder in the Server Explorer or the Eclipse Explorers that contains the task that you want to modify.
3. Do one of the following:
  - Double click on a task or sub-task in the task tree or list.The **Properties** dialog box opens.
4. Select the **Resources** tab and click **Add**.  
The **Select Task Resources** dialog box opens.
5. Select the team members you want to assign to the task in the **Users** list, and click **Add**.  
The selected names are moved from the **Users** list to the **Assigned Resources** list.
6. Click **OK**.

## Estimating Tasks

The task component includes a **Time** tab on which you can record the amount of time needed to complete a task or sub-task.

To enter the estimated time to complete a task

1. Open the **Task** view and click **Link with Selection**.
2. Select the folder in the Server Explorer or the Eclipse Explorers that contains the task that you want to modify.
3. Do one of the following:
  - Double click on a task or sub-task in the task tree or list.The **Properties** dialog box opens.
4. Click the **Time** tab.
5. Use the **Start** and **Finish** buttons to select a start and finish date.
6. Type the estimated hours required to complete the task in the **Work** field.  
The rest of the **Time** pane is disabled, however, the values for **Actual** and **Variance** automatically calculate when the **Work** value changes.
7. Click **OK**.

## Adding Notes to Tasks

When you use the task component, you can enter additional information about the task or sub-task on the **Notes** tab in the **Task Properties** dialog box.

1. Open the **Task** view and click **Link with Selection**.
2. Select the folder in the Server Explorer or the Eclipse Explorers that contains the task that you want to modify.
3. Do one of the following:
  - Double click on a task or sub-task in the task tree or list.

The **Properties** dialog box opens.

4. Click the **Notes** tab and type notes about the task in the **Notes** field.
5. Click **OK**.

## Working with Work Records in Tasks

After working on a task or subtask, you should add a work record to indicate what was done and the time spent. For example, if you work on a task for one hour on Day one and for three hours on Day two, you would enter two work records, one for each day. You can edit or delete previously entered work records



**Important:** After you have added a work record to a task, you cannot create subtasks for that task.

### Adding a Work Record to a Task

1. Open the **Task** view and click **Link with Selection**.
2. Select the folder in the Server Explorer or the Eclipse Explorers that contains the task that you want to modify.
3. In the **Task** view, double click on a task or sub-task in the task tree or list. The **Properties** dialog box opens.
4. Select the **Work** tab, and click **Add**. The **Work Record** dialog box opens and displays your **User Name** in the list.
5. Click the **Date** button and select a date for the work record.
6. Type the number of hours worked in the **Work** field.
7. Type the number of hours it will take to complete the task in the **Remaining Work** field.
8. Type comments about the progress that has been made in the **Comments** field.
9. Click **OK**.

### Editing a Work Record for a Task

1. Open the **Task** view and click **Link with Selection**.
2. Select the folder in the Server Explorer or the Eclipse Explorers that contains the task that you want to modify.
3. Do one of the following:
  - Double click on a task or sub-task in the task tree or list.The **Task Properties** dialog box displays.
4. On the **Work** page, select a record from the **Work Records** list and click **Edit**. This opens the **Work Records** dialog box.
5. In the **Work Records** dialog box, make any changes to the work record.



6. Click **OK**.

## Deleting a Work Record from a Task

1. Open the **Task** view and click **Link with Selection**.
2. Select the folder in the Server Explorer or the Eclipse Explorers that contains the task that you want to modify.
3. Do one of the following:
  - Double click on a task or sub-task in the task tree or list.The **Properties** dialog box opens.
4. On the **Work** page, select a record from the **Work Records** list and click **Delete**.
5. The message, Delete Work Record? appears. Click **Yes** to confirm the deletion.
6. Click **OK**.

## Sample Task Detail View Template

You can use the following sample for customizing the **Detail** view for the task component.

```
<html>
<head></head>
<body>
<b><b>MS WBS Code:</b> ~StTaskWBSCode~~<br>
<b>Attention Notes:</b> ~StTaskAttentionNotes~~<br>
<b>Estimated Start:</b> ~StTaskEstimatedStart~~
<br>
<b>My Lock:</b> ~MyLock~~ <br>
<b>Folder Path:</b> ~Folder Path~~<br>
<b>Estimated Hours Variance:</b> ~StTaskEstimatedHoursVariance~~ <br>
<b>Task Duration:</b> ~StTaskDuration~~<br>
<b>Version:</b> ~RevisionNumber~~<br>
<b>Resource IDs:</b> ~StTaskResourceIDs~~<br>
<b>Flag:</b> ~Flag~~ <br>
<b>Short Comment:</b> ~ShortComment~~<br>
<b>Created By:</b> ~CreatedUserID~~<br>
<b>Responsibility:</b> ~StTaskResponsibility~~<br>
<b>Constraint Date:</b> ~StTaskConstraintDate~~<br>
<b>Created Time:</b> ~CreatedTime~~<br>
<b>Share State:</b> ~ShareState~~<br>
<b>Locked By:</b> ~ExclusiveLocker~~<br>
<b>Priority:</b> ~StTaskPriority~~<br>
<b>Resource Count:</b> ~StTaskResourceCount~~<br>
<b>Estimated Finish:</b> ~StTaskEstimatedFinish~~<br>
<b>Actual Start:</b> ~StTaskActualStart~~<br>
<b>Actual Finish:</b> ~StTaskActualFinish~~<br>
<b>Estimated Hours:</b> ~StTaskEstimatedHours~~<br>
<b>Is My Task?:</b> ~StTaskIsMyTask~~<br>
<b>Attachment IDs:</b> ~AttachmentIDs~~<br>
<b>Estimated Start Variance:</b> ~StTaskEstimatedStartVariance~~ <br>
<b>Deleted By:</b> ~DeletedUserID~~<br>
<b>Dot Notation:</b> ~DotNotation~~<br>
<b>Parent Task ID:</b> ~StTaskParentID~~<br>
<b>MS Task Unique ID:</b> ~StTaskUniqueID~~<br>
<b>Status:</b> ~StTaskStatus~~<br>
<b>Notes:</b> ~StTaskNotes~~ <br>
<b>Read Status:</b> ~ReadStatus~~<br>
<b>Children Count:</b> ~ChildrenCount~~<br>
<b>Constraint Type:</b> ~StTaskConstraintType~~<br>
<b>Last Work/Dependency Update:</b> ~StWorkDependencyLastUpdate~~ <br>
<b>Modified By:</b> ~ModifiedUserID~~<br>
<b>New Revision Comment:</b> ~NewRevisionComment~~ <br>
```

```

<b>Non-Exclusive Lockers:</b> ~~NonExclusiveLockers~~ <br>
<b>Resource Names:</b> ~~StTaskResourceNames~~<br>
<b>Read Status User List:</b> ~~ReadStatusUserList~~ <br>
<b>Task Type:</b> ~~StTaskType~~<br>
<b>Estimated Finish Variance:</b> ~~StTaskEstimatedFinishVariance~~ <br>
<b>Actual Hours:</b> ~~StTaskActualHours~~<br>
<b>Revision Flags:</b> ~~RevisionFlags~~<br>
<b>Percent Complete:</b> ~~StTaskPercentComplete~~<br>
<b>Task Name:</b> ~~StTaskName~~<br>
<b>Attachment Count:</b> ~~AttachmentCount~~<br>
<b>CommentID:</b> ~~CommentID~~<br>
<b>Is Replicated:</b> ~~Is Replicated~~<br>
<b>Flag User List:</b> ~~FlagUserList~~<br>
<b>Object ID:</b> ~~ID~~ <br>
<b>Task Origin:</b> ~~StTaskOrigin~~<br>
<b>Modified Time:</b> ~~ModifiedTime~~<br>
<b>MS Project File Name:</b> ~~StTaskMSProjectFileName~~ <br>
<b>Deleted Time:</b> ~~DeletedTime~~<br>
<b>Milestone:</b> ~~StTaskMilestone~~<br>
<b>Work Record Count:</b> ~~WorkRecCount~~<br>
<b>Configuration Time:</b> ~~ConfigurationTime~~<br>
<b>Last MS Project Update:</b> ~~StTaskMSProjectLastUpdate~~ <br>
<b>MS Task GUID:</b> ~~StTaskGUID~~<br>
<b>End Modified Time:</b> ~~EndModifiedTime~~<br>
<b>Task Number:</b> ~~StTaskNumber~~<br>
<b>Needs Attention:</b> ~~StTaskNeedsAttention~~<br>
<b>Attachment names:</b> ~~AttachmentNames~~<br>
<b>Comment:</b> ~~Comment~~ <br>
<b>Read Only:</b> ~~ReadOnly~~ <br>
</body>
</html>


```

## Fields Used in Detail View Templates


The fields used in the Detail view HTML templates are recognized by the client when they are contained between double tilde ~~ characters. For example: `~~StTaskEstimatedStart~~` represents the estimated start date field found in the **Task Properties** dialog box. Refer to the link at the bottom of this topic for more information about the fields that you can use in the Detail view templates.

## Task Fields

This section lists all the task fields in alphabetical order.

 **Note:** Client-calculated fields cannot be used in custom email notifications or the StarTeam Notification Agent. Reports can use any field name.

- |                      |  |
|----------------------|--|
| <b>Actual Finish</b> | <p>Values: date/time.</p> <p>Internal Identifier: <code>StTaskActualFinish</code>.</p> <p>The actual finish date for a task.</p>         |
| <b>Actual Hours</b>  | <p>Values: number.</p> <p>Internal Identifier: <code>StTaskActualHours</code>.</p> <p>The number of hours spent completing the task.</p> |
| <b>Actual Start</b>  | <p>Values: date/time.</p> <p>Internal Identifier: <code>StTaskActualStart</code>.</p> <p>The actual start date for a task.</p>           |

<b>Attachment Count</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>AttachmentCount</code>.</p> <p>The number of files attached to an item.</p>
<b>Attachment IDs (Advanced)</b>	<p>Values: <code>byte array</code>. Displayed as a bracketed series of numbers in hex format. For example: <code>[00 00 00 00 02 00 00 00]</code> indicates two specific attachments.</p> <p>Internal Identifier: <code>AttachmentCount</code>.</p> <p>Cannot be used in queries. The ID numbers assigned to attachments. For example, the first attachment within a project is <code>00 00 00 00</code>.</p>
<b>Attachment Names</b>	<p>Values: <code>text</code> containing a series of file names separated by spaces.</p> <p>Internal Identifier: <code>AttachmentNames</code>.</p> <p>The names of the files attached to an item.</p>
<b>Attention Notes</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>StTaskAttentionNotes</code>.</p> <p>The text in the <b>Needs Attention</b> note.</p>
<b>Children Count</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>ChildrenCount</code>.</p> <p>The number of items that are children of this item. This is a client-calculated field.</p>
<b>Comment</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>Comment</code>.</p> <p>The initial 2000 characters provided as the reason for changing an item's properties or contents are stored in the <b>Short Comment</b> field. The <b>Comment</b> field stores those 2000 characters and any additional text. Changing an item's properties causes the application to create a new revision.</p> <p> <b>Note:</b> To include a <b>Link</b> comment, the <b>Comment</b> field is the value to use in an HTML report.</p>
<b>CommentID (Advanced)</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>CommentID</code>.</p> <p>The ID number assigned to the revision comment. Displays <code>-1</code> if no revision comment was supplied.</p>
<b>Configuration Time</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>ConfigurationTime</code>.</p> <p>Indicates the time to which an item is configured. If you configure an item to a specific time, this field contains that time. If you configure an item to a label or promotion state, this field shows either the time at which the label was created or the time at which the label associated with the promotion state was created.</p>
<b>Constraint Date</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>StTaskConstraintDate</code>.</p> <p>A task's constraint date from MS Project.</p>

<b>Constraint Type</b>	<p>Values: As Late As Possible, As Soon As Possible, Finish No Earlier Than, Finish No Later Than, Must Finish On, Must Start On, Start No Earlier Than, Start No Later Than.</p> <p>Internal Identifier: StTaskConstraintType.</p> <p>A task's constraint type from MS Project.</p>
<b>Created By</b>	<p>Values: list of users, &lt;None&gt;.</p> <p>Internal Identifier: CreatedUserID.</p> <p>The name of the user who created the first revision in the view. This is either the user who added the item to the project, or the user who checked in the revision that branched.</p>
<b>Created Time</b>	<p>Values: date/time.</p> <p>Internal Identifier: CreatedTime</p> <p>The time at which the first revision in the view was created.</p>
<b>Deleted By</b>	<p>Values: list of users, &lt;None&gt;.</p> <p>Internal Identifier: DeletedUserID.</p> <p>The name of the user who deleted the item. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.</p>
<b>Deleted Time</b>	<p>Values: date/time.</p> <p>Internal Identifier: DeletedTime.</p> <p>The time at which an item was deleted. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.</p>
<b>Dot Notation</b>	<p>Values: text.</p> <p>Internal Identifier: DotNotation.</p> <p>The branch revision number, for example, 1.2.1.0.</p>
<b>End Modified Time (Advanced)</b>	<p>Values: date/time.</p> <p>Internal Identifier: EndModifiedTime.</p> <p>The date and time at which a revision ceased to be the tip revision. Although this field can be displayed in the upper pane, its value is always blank. This is because, at any given configuration time, the item is still the tip revision.</p>
<b>Estimated Finish</b>	<p>Values: date/time.</p> <p>Internal Identifier: StTaskEstimatedFinish.</p> <p>The estimated finish date for a task.</p>
<b>Estimated Finish Variance</b>	<p>Values: date/time.</p> <p>Internal Identifier: StTaskEstimatedFinishVariance.</p> <p>The difference between the estimated and the actual finish date for a task.</p>
<b>Estimated Hours</b>	<p>Values: number</p> <p>Internal Identifier: StTaskEstimatedHours.</p>

	The number of hours spent completing the task.
<b>Estimated Hours Variance</b>	<p>Values: <i>number</i>.</p> <p>Internal Identifier: <i>StTaskEstimatedHoursVariance</i></p> <p>The difference between the estimated and the actual number of hours spent completing the task.</p>
<b>Estimated Start</b>	<p>Values: <i>date/time</i>.</p> <p>Internal Identifier: <i>StTaskEstimatedStart</i>.</p> <p>The estimated start date for a task.</p>
<b>Estimated Start Variance</b>	<p>Values: <i>date/time</i>.</p> <p>Internal Identifier: <i>StTaskEstimatedStartVariance</i>.</p> <p>The difference between the estimated and the actual start date for a task.</p>
<b>Flag</b>	<p>Values: <i>No, Yes</i>.</p> <p>Internal Identifier: <i>Flag</i>.</p> <p>Marks or bookmarks files in the upper pane on your workstation. This is a client-calculated field.</p>
<b>Flag User List (Advanced)</b>	<p>Values: text displayed as a list of user names. For example: [<i>Greg, Sam</i>] indicates user names.</p> <p>Internal Identifier: <i>FlagUserList</i>.</p> <p>Can be used in queries. Identifies users who have set flags on a given item.</p>
<b>Folder Path</b>	<p>Values: <i>text</i>.</p> <p>Internal Identifier: <i>Folder Path</i> (contains spaces).</p> <p>The path to the folder. This is not the path to the working folder.</p>
<b>Is My Task?</b>	<p>Values: <i>No, Yes</i>.</p> <p>Internal Identifier: <i>IsMyTask?</i>.</p> <p>Indicates whether the logged on user is responsible for a task. This is a client-calculated field.</p>
<b>Is Replicated</b>	<p>Values: <i>0, 1</i>.</p> <p>Internal Identifier: <i>Is Replicated</i> (contains spaces).</p> <p>Indicates whether the task is from MS Project task.</p>
<b>Item Deleted By</b>	<p>Values: list of users, <i>None</i>.</p> <p>Internal Identifier: <i>ItemDeletedUserID</i>.</p> <p>The name of the user who deleted this item. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.</p>
<b>Item Deleted Time</b>	<p>Values: <i>date/time</i>.</p> <p>Internal Identifier: <i>ItemDeltedTime</i>.</p> <p>The time at which the item was deleted. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.</p>

<b>Last MS Project Update</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>StTaskMSProjectLastUpdate</code>.</p> <p>The date that a task was last updated from MS Project.</p>
<b>Last Work/Dependency Update</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>StWorkDependencyLastUpdate</code>.</p> <p>The last time that a work record or a dependency (task successor or predecessor) was added, edited, or deleted. This field is for use with MS Project.</p>
<b>Locked By</b>	<p>Values: list of users, <code>&lt;None&gt;</code>.</p> <p>Internal Identifier: <code>ExclusiveLocker</code>.</p> <p>The name of the user who has exclusively locked a folder.</p>
<b>Milestone</b>	<p>Values: <code>No</code>, <code>Yes</code>.</p> <p>Internal Identifier: <code>StTaskMilestone</code></p> <p>Indicates whether a task represents a milestone. In MS Project, the definition for a milestone is a task of zero time length. It serves as a heading for one or more tasks to which a time length has been assigned.</p> <p>In the application, a task has a milestone check box. After work is assigned to a task, it is no longer a milestone.</p>
<b>Modified By</b>	<p>Values: list of users, <code>&lt;None&gt;</code>.</p> <p>Internal Identifier: <code>ModifiedUserID</code>.</p> <p>The name of the user who last modified the item.</p>
<b>Modified Time</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>ModifiedTime</code>.</p> <p>The time at which an item was last modified. The item may have been checked in or had its properties changed. For folders, this has nothing to do with the working folder. Use <b>Local Time Stamp</b> for the time a working folder was last modified.</p>
<b>MS Project File Name (Advanced)</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>StTaskMSProjectFileName</code>.</p> <p>The name of the MS project file from which a task was exported.</p>
<b>MS Task GUID (Advanced)</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>StTaskGUID</code>.</p> <p>The GUID for a task in MS Project.</p>
<b>MS Task Unique ID (Advanced)</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>StTaskUniqueID</code>.</p> <p>The unique ID for a task in MS Project.</p>
<b>MS WBS Code (Advanced)</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>StTaskWBSCode</code>.</p> <p>A task's WBS code from MS Project.</p>

<b>My Lock</b>	<p>Values: Exclusively Locked By Me, Non-exclusively Locked By Me, Not Locked By Me.</p> <p>Internal Identifier: MyLock.</p> <p>Indicates whether the current user has the item locked and, if so, whether that lock is exclusive or not. This is a client-calculated field.</p>
<b>Needs Attention</b>	<p>Values: No, Yes.</p> <p>Internal Identifier: StTaskNeedsAttention.</p> <p>Indicates that the check box for <b>Needs Attention</b> has been selected.</p>
<b>New Revision Comment (Advanced)</b>	<p>Values: text.</p> <p>Internal Identifier: NewRevisionComment.</p> <p>Internal use only. The client uses this value during the item update process. The field always appears empty if added to the upper pane. This is a client-calculated field.</p>
<b>Non-Exclusive Lockers</b>	<p>Values: text.</p> <p>Internal Identifier: NonExclusiveLockers.</p> <p>The names of the users who have locked the folder non-exclusively.</p>
<b>Notes</b>	<p>Values: text.</p> <p>Internal Identifier: Notes.</p> <p>Text comments on the effort levels for this item.</p>
<b>Object ID</b>	<p>Values: number.</p> <p>Internal Identifier: ID.</p> <p>Each item is assigned an object ID when it is added to a view. For applicable items, when it is branched in a child view, it is assigned another object ID. The original ID belongs to the folder in the parent view.</p>
<b>Parent ID (Advanced)</b>	<p>Values: number.</p> <p>Internal Identifier: ParentID.</p> <p>The object ID of an item in the parent view. The Parent ID is -1 if this view has no parent view.</p>
<b>Percent Complete</b>	<p>Values: number.</p> <p>Internal Identifier: StTaskPercentComplete.</p> <p>A percentage indicating how much of a task has been completed.</p>
<b>Priority</b>	<p>Values: Do Not Level, High, Higher, Highest, Low, Lower, Lowest, Medium, Very High, Very Low.</p> <p>Internal Identifier: StTaskPriority.</p> <p>Indicates the priority given to a task. These priorities are identical to those in MS Project.</p>
<b>Read Only (Advanced)</b>	<p>Values: No, Yes.</p> <p>Internal Identifier: ReadOnly.</p>

Indicates whether the item's configuration is read-only (as in a rollback configuration of a view)/its behavior does not allow it to branch on modification. For folders, do not confuse a read-only configuration (an application issue) with a read-only folder (an operating system issue). A read-only folder cannot be edited and saved to disk. A folder whose configuration is read-only can be edited and saved to disk; it just cannot be checked in.

**Read Status**

Values: `Read`, `Unread`.

Internal Identifier: `ReadStatus`.

Indicates whether an item is considered read or not read. This is a client-calculated field.

**Read Status User List**

Values: `text` displayed as a list of user names. For example: `[Greg, Sam]` indicates user names.

Internal Identifier: `ReadStatusUserList`.

Can be used in queries. Identifies users for whom a given item's status is **Unread**.

**Resource Count**

Values: `number`.

Internal Identifier: `StTaskResourceCount`.

The number of users listed as resources for a task.

**Resource IDs (Advanced)**

Values: `byte array`, displayed as a bracketed series of numbers in hex format. For example, `[14 00 00 00]` indicates a specific user.

Internal Identifier: `StTaskResourceIDs`.

Cannot be used in queries. The ID numbers assigned to the users who are this task's resources.

**Resource Names**

Values: `text` containing a series of user names separated by spaces

Internal Identifier: `StTaskResourceNames`

The names of the users who are this task's resources.

**Responsibility**

Values: list of users, `<None>`.

Internal Identifier: `StTaskResponsibility`.

The name of the user who is currently responsible for the task.

**Revision Flags (Advanced)**

Values: `0`.

Internal Identifier: `RevisionFlags`.

Internal use only.

**Share State**

Values: `DerivedShare`, `Not Shared`, `Root Share`.

Internal Identifier: `ShareState`

Indicates whether this item is shared. `Not Shared` means that the item is not shared. `Root Share` means that the item is shared and this item is the original (or root) reference. `DerivedShare` means that the item is shared, but this item is not the original (or root) reference.

**Short Comment**

Values: `text`.

Internal Identifier: `ShortComment`.



	Stores the initial 2000 characters provided as the reason for changing an item's properties or contents. Additional text is stored in the <b>Comment</b> field.
<b>Status</b>	<p>Values: <code>Closed</code>, <code>Finish</code>, <code>Hold</code>, <code>In Progress</code>, <code>Pending</code>, <code>Ready To Start</code>.</p> <p>Internal Identifier: <code>STTaskStatus</code>.</p> <p>Indicates the status of the task.</p>
<b>Task Duration</b>	<p>Values: number</p> <p>Internal Identifier: <code>STTaskDuration</code></p> <p>The number of hours during which any user is working on a task. For example if two people will work eight hours on a task, the duration is eight hours if they work at the same time or a maximum of 16 hours if they do the work on different days.</p>
<b>Task Name</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>STTaskName</code>.</p> <p>The name of the task.</p>
<b>Task Number</b>	<p>Values: number.</p> <p>Internal Identifier: <code>StTaskNumber</code>.</p> <p>The number assigned to a task. For example, if the <b>Object ID</b> is 0, the task number is 1.</p>
<b>Task Origin</b>	<p>Values: <code>MSPProject</code>, <code>StarTeam</code>.</p> <p>Internal Identifier: <code>STTaskOrigin</code>.</p> <p>Indicates whether the task was created in the application or exported to the application from Microsoft Project.</p>
<b>Task Type</b>	<p>Values: <code>Fixed Duration</code>, <code>Fixed Units</code>, <code>Fixed Work</code>.</p> <p>Internal Identifier: <code>StTaskType</code>.</p> <p>A task's type in MS Project.</p>
<b>Version (Advanced)</b>	<p>Values: number.</p> <p>Internal Identifier: <code>RevisionNumber</code>.</p> <p>The last number in the branch revision number. For example, if the branch revision number is 1.3.1.2, the version is 2.</p>
<b>Work Record Count</b>	<p>Values: number.</p> <p>Internal Identifier: <code>WorkRecCount</code>.</p> <p>The number of work records currently added to a task.</p>

## Task Properties

This topic presents the task properties and their descriptions as displayed in the **Task Properties** dialog box. The **Task Properties** dialog box contains the following tabbed pages of properties.

### Task

The following properties are on the **Task** tab.

<b>Subtask Of</b>	Displays the name of the task for which this item is a subtask (if this item is a subtask).
<b>Name</b>	Displays the name of the task or subtask.
<b>Responsibility</b>	Displays the name of the person responsible for the completion of this task or subtask. Other people can be assigned as additional resources.
<b>Milestone</b>	Indicates that the task or subtask should be treated as a milestone.
<b>Status</b>	Displays the task status: <ul style="list-style-type: none"> <li><b>Pending</b>            Waiting for completion of a predecessor task.</li> <li><b>Ready To Start</b>    Work can be started on the task.</li> <li><b>In Progress</b>        Work has been entered for the task.</li> <li><b>Finished</b>            Work is finished on the task.</li> <li><b>Closed</b>             Task is completed and closed.</li> <li><b>Hold</b>                Work temporarily stopped on the task, usually to wait for completion of another task.</li> </ul>
<b>Priority</b>	Displays the task priority level. The default is <b>Medium</b> . These priorities are identical to those in MS Project. <b>Do Not Level</b> is a Microsoft Project-specific term you should ignore.
<b>Duration</b>	Indicates the number of hours expected for completion of the task.
<b>Percent Complete</b>	Displays the percentage of work that has been completed on a task.
<b>Needs Attention</b>	Notifies team leaders or task reviewers that this task requires attention. Enter the information about why this task needs attention in the text box below the <b>Needs Attention</b> check box.

## Resources

The **Resources** tab lists the task resources. You can assign responsibility to team members by adding them to the list with the **Add** button, and you can remove them using the **Remove** button.

## Time

The following properties are on the **Time** tab.

<b>Plan Start</b>	Displays the start date for the task.
<b>Plan Finish</b>	Displays the finish date for the task.
<b>Plan Work</b>	Indicates the number of hours estimated to complete this task.
<b>Actual Start</b>	Displays the actual start date calculated from work record entry.
<b>Actual Finish</b>	Displays the actual finish date task status changes to <b>Finished</b> .
<b>Actual Work</b>	Indicates the actual number of hours taken to complete the task, calculated from Work Records.
<b>Variance Start</b>	Displays the variance in days between expected start date and actual start date. This is read-only calculated value.
<b>Variance Finish</b>	Displays the variance in days between expected finish date and actual finish date. This is read-only calculated value.

**Variance Work** Displays the variance in number of hours between estimated and actual duration. This is read-only calculated value.

## Work

The **Work** tab lists all the work records entered for this task. Each work record has the following properties.

<b>User Name</b>	Displays the name of person who performed the work for this work record entry.
<b>Date</b>	Displays the date of work record entry.
<b>Work Hours</b>	Indicates the number of hours worked for this work record entry.
<b>Remaining Work</b>	Indicates the remaining number of hours left to complete the task.
<b>Comments</b>	Displays text comments explaining what work was done for this work record.
<b>Total Actual Work</b>	Displays read-only calculated field of the total time spent on this task based on the work records entered.

## Notes

The **Notes** tab is a simply a text box for capturing notes about the task.

## Custom

You can create custom properties for an item which will display in the item **Properties** dialog box.

The following properties are on the **Custom** tab.

<b>Property</b>	Displays each custom property name.
<b>Value</b>	Displays the values for each custom property. Double-click the property name to edit the value.

## Attachments

The **Attachments** tab contains a list of all the files attached to the current item.

## Comment

The following properties are on the **Comment** tab.

<b>Comment For This Revision</b>	Displays the reason for the changes to the current revision.
<b>Comment For New Revision</b>	Displays the reason for the changes to the new revision.

# Topics

*Topics* are threaded conversations, that is, a series of messages that indicate how the messages are related. Each series of messages forms a tree with the initial message at its root. The topic component provides threaded conversations that you can place in specific project folders and link to specific project items. For example, you can link a topic to the change requests and file revisions that result from the topic discussion.

The **Topic** view consists of topics and a series of responses to each topic. A series of topic trees are eventually formed, each of which consists of a root topic and its responses. The topic tree resembles a conversation that may go on among several people. In the client, this is called a threaded conversation because a topic and its responses are threaded together, starting with the root topic. By reading each response in a thread, one after the other, and the responses to those responses, you can see how the discussion has evolved. A number of other operations can be performed on topics or responses such as moving or sharing them.

## Historical Value of Topics

Topics can raise general questions about the project or start very specific discussions about issues, such as feature implementation. While the responses can lead to resolution of these issues, the historical value of these conversations to the project can be even more significant. Future team members can:

- Reassess decisions more capably.
- Avoid retrying solutions that were previously found faulty.
- Understand why a particular solution to a problem became necessary and, therefore, not replace that solution with one that does not meet all the necessary criteria.

## How Topics Can Help

Any type of threaded messaging improves teamwork on product development. However, because StarTeam has tightly integrated components, it enables team members to:

- Search topics and responses for specific words or phrases.
- Sort topics and responses.
- Filter topics and responses.
- View relationships between topics and their responses.
- Move and share topics (from the tree format).
- Link topics directly to folders or other items, such as change requests.
- Ask questions and quickly receive input while working on a file.
- Attach notes to a topic explaining why a particular method was used.
- Point out aspects of the project that may need to change in a later release.

# Creating Topics

To start a threaded conversation, you must first create a topic.

1. Open the **Topic** view, and click **Link with Selection**.
2. Select the folder in the Server Explorer or the Eclipse Explorers that contains the task that you want to modify.
3. Right-click inside the **Topic** view and choose **New...** The **New Topic** dialog box opens.
4. Click the **Topic** tab and type the title of your topic in the **Title** field.
5. Type the content for this topic in the **Content** field.
6. Use the **Options** tab if you want to send the topic to specific team members, assign a priority to the topic, or indicate a status for the topic.
  1. Select the **Options** tab.
  2. Click **Add**. The **Select Topic Recipients** dialog box opens.
  3. Select the team members from the list, then click **Add**.
  4. To assign a priority to the topic, select **Low**, **Normal**, or **High** from the **Priority** list.
  5. To specify a topic status, select either **Active** or **Inactive** from the **Status** list. The default status is **Active**.
7. If your administrator created additional topic properties, you can access them on the **Custom** tab.
  1. Double-click a custom property on the **Custom** tab to open the **Edit Property** dialog box.
  2. Type or select a new value for the property by double-clicking on the field:

<b>integer, text, and real fields</b>	<b>Value</b> is a text box.
<b>enumerated types and user IDs</b>	<b>Value</b> is a list box.

### dates and times

**Value** has a **Date** check box and a **Time** check box, each of which is followed by a date or time in the format for your locale.



**Tip:** To enter a blank value for a **GroupList** or **UserList** property, click on a selected row to deselect it. When the item is no longer highlighted, click **OK**.

3. Click **Apply**.
8. Use the **Attachments** tab if you want to attach a file, note, or graphic with your topic.
9. Optionally, select the **Comment** tab to add additional notes or a comment in the **Comment for new revision** field.
10. Click **OK**. This action enters the new topic in the upper pane of the **Topics** component.

If the tree format is selected, the topic title, your user name, and the time stamp display. If the list format is selected, the list displays the same information, but includes one additional column, **Description**, which shows the first few words in the topic text.

## Responding to Topics or Responses

After someone starts a topic, you can reply to the topic or to one or more of its responses.

1. Open the **Topic** view, and click **Link with Selection**.
2. Select the folder in the Server Explorer or the Eclipse Explorers that contains the task that you want to modify.
3. Right click the topic and select **Respond**. The **New Topic** dialog box opens.
4. On the **Topic** tab in the **New Topic** dialog box, type the title of your response in the **Title** field.
5. Type your remarks in the **Content** field.
6. Use the **Options** tab if you want to send the topic to specific team members, assign a priority to the topic, or indicate a status for the topic.
  1. Select the **Options** tab.
  2. Click **Add**. The **Select Topic Recipients** dialog box opens.
  3. Select the team members from the list, then click **Add**.
  4. To assign a priority to the topic, select **Low**, **Normal**, or **High** from the **Priority** list.
  5. To specify a topic status, select either **Active** or **Inactive** from the **Status** list. The default status is **Active**.
7. If your administrator created additional topic properties, you can access them on the **Custom** tab.
  1. Double-click a custom property on the **Custom** tab to open the **Edit Property** dialog box.
  2. Type or select a new value for the property by double-clicking on the field:

**integer, text, and real fields**      **Value** is a text box.

**enumerated types and user IDs**      **Value** is a list box.

### dates and times

**Value** has a **Date** check box and a **Time** check box, each of which is followed by a date or time in the format for your locale.



**Tip:** To enter a blank value for a **GroupList** or **UserList** property, click on a selected row to deselect it. When the item is no longer highlighted, click **OK**.

3. Click **Apply**.
8. Use the **Attachments** tab if you want to attach a file, note, or graphic with your topic.
9. Optionally, select the **Comment** tab to add additional notes or a comment in the **Comment for new revision** field.
10. Click **OK**. This action enters the new response as a child of the topic.

If the tree format is selected, the response appears in relation to other parts of the threaded conversation. The response title, your user name, and the time stamp also display. If the list format is selected, the list

displays the same information, but includes one additional column, **Description**, which shows the first few words in the response text.

## Sample Topic Detail View Template

Use the following sample for customizing the **Detail** view for the topic component.

Cut and paste the HTML contents below into a file named `topic.details.html`, and place it in the `Application Data\Micro Focus\StarTeam` folder for the current user. For example, on a Microsoft Windows system, this file would be located in the `C:\Documents and Settings\USER\Application Data\Micro Focus\StarTeam` folder.

```
<html>
<head></head>
<body>
<b>Flag User List</b>: ~~FlagUserList~~<br>
<b>Version</b>: ~~RevisionNumber~~<br>
<b>Status</b>: ~~Status~~<br>
<b>Read Status User List</b>: ~~ReadStatusUserList~~<br>
<b>Modified Time</b>: ~~ModifiedTime~~<br>
<b>Attachment IDs</b>: ~~AttachmentIDs~~<br>
<b>Created Time</b>: ~~CreatedTime~~<br>
<b>Content</b>: ~~Description~~<br>
<b>Share State</b>: ~~ShareState~~<br>
<b>CommentID</b>: ~~CommentID~~<br>
<b>Created By</b>: ~~CreatedUserID~~<br>
<b>Deleted Time</b>: ~~DeletedTime~~<br>
<b>Children Count</b>: ~~ChildrenCount~~<br>
<b>Title</b>: ~~Title~~<br>
<b>Non-Exclusive Lockers</b>: ~~NonExclusiveLockers~~<br>
<b>Topic Number</b>: ~~TopicNumber~~<br>
<b>Recipient IDs</b>: ~~RecipientIDs~~<br>
<b>Short Comment</b>: ~~ShortComment~~<br>
<b>Recipient Count</b>: ~~RecipientCount~~<br>
<b>Locked By</b>: ~~ExclusiveLocker~~<br>
<b>Folder Path</b>: ~~Folder Path~~<br>
<b>Object ID</b>: ~~ID~~<br>
<b>Flag</b>: ~~Flag~~<br>
<b>Recipient Names</b>: ~~RecipientNames~~<br>
<b>Read Only</b>: ~~ReadOnly~~<br>
<b>My Lock</b>: ~~MyLock~~<br>
<b>Configuration Time</b>: ~~ConfigurationTime~~<br>
<b>Comment</b>: ~~Comment~~<br>
<b>Revision Flags</b>: ~~RevisionFlags~~<br>
<b>End Modified Time</b>: ~~EndModifiedTime~~<br>
<b>Am I Recipient?</b>: ~~AmIRecipient~~<br>
<b>New Revision Comment</b>: ~~NewRevisionComment~~<br>
<b>Attachment Count</b>: ~~AttachmentCount~~<br>
<b>Type</b>: ~~Type~~<br>
<b>Priority</b>: ~~Priority~~<br>
<b>Modified By</b>: ~~ModifiedUserID~~<br>
<b>Deleted By</b>: ~~DeletedUserID~~<br>
<b>Attachment names</b>: ~~AttachmentNames~~<br>
<b>Dot Notation</b>: ~~DotNotation~~<br>
<b>Read Status</b>: ~~ReadStatus~~<br>
<b>Parent Topic ID</b>: ~~ParentTopicID~~<br>
</body>
</html>
```

### Fields Used in Detail View Templates

The fields used in the Detail view HTML templates are recognized by the client when they are contained between double tilde `~~` characters. For example: `~~Status~~` represents the represents the Status field

found in the field found in the **Topic Properties** dialog box. Refer to the link at the bottom of this topic for more information about the fields that you can use in the Detail view templates.

## Topic Fields

This section lists all the fields in alphabetical order.



**Note:** Client-calculated fields cannot be used in custom email notifications or Notification Agent. Reports can use any field name.

### Am I Recipient?

Values: `No`, `Yes`.

Internal Identifier: `AmIRecipient?`.

Indicates whether the logged on user is a recipient of a topic. This is a client-calculated field.

### Attachment Count

Values: `number`.

Internal Identifier: `AttachmentCount`.

The number of files attached to an item.

### Attachment IDs (Advanced)

Values: `byte array`. Displayed as a bracketed series of numbers in hex format. For example: `[00 00 00 00 02 00 00 00]` indicates two specific attachments.

Internal Identifier: `AttachmentCount`.

Cannot be used in queries. The ID numbers assigned to attachments. For example, the first attachment within a project is `00 00 00 00`.

### Attachment Names

Values: `text` containing a series of file names separated by spaces.

Internal Identifier: `AttachmentNames`.

The names of the files attached to an item.

### Children Count

Values: `number`.

Internal Identifier: `ChildrenCount`.

The number of items that are children of this item. This is a client-calculated field.

### Comment

Values: `text`.

Internal Identifier: `Comment`.

The initial 2000 characters provided as the reason for changing an item's properties or contents are stored in the **Short Comment** field. The **Comment** field stores those 2000 characters and any additional text. Changing an item's properties causes the application to create a new revision.



**Note:** To include a **Link** comment, the **Comment** field is the value to use in an HTML report.

### CommentID (Advanced)

Values: `number`.

Internal Identifier: `CommentID`.

The ID number assigned to the revision comment. Displays `-1` if no revision comment was supplied.

### Configuration Time

Values: `date/time`.

Internal Identifier: `ConfigurationTime`.

Indicates the time to which an item is configured. If you configure an item to a specific time, this field contains that time. If you configure an item to a label or promotion state, this field shows either the time at which the label was created or the time at which the label associated with the promotion state was created.

**Content**

Values: `text`.

Internal Identifier: `Description`.

The text of a topic.

**Created By**

Values: list of users, `<None>`.

Internal Identifier: `CreatedUserID`.

The name of the user who created the first revision in the view. This is either the user who added the item to the project, or the user who checked in the revision that branched.

**Created Time**

Values: `date/time`.

Internal Identifier: `CreatedTime`

The time at which the first revision in the view was created.

**Deleted By**

Values: list of users, `<None>`.

Internal Identifier: `DeletedUserID`.

The name of the user who deleted the item. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.

**Deleted Time**

Values: `date/time`.

Internal Identifier: `DeletedTime`.

The time at which an item was deleted. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.

**Dot Notation**

Values: `text`.

Internal Identifier: `DotNotation`.

The branch revision number, for example, `1.2.1.0`.

**End Modified Time (Advanced)**

Values: `date/time`.

Internal Identifier: `EndModifiedTime`.

The date and time at which a revision ceased to be the tip revision. Although this field can be displayed in the upper pane, its value is always blank. This is because, at any given configuration time, the item is still the tip revision.

**Flag**

Values: `No`, `Yes`.

Internal Identifier: `Flag`.

Marks or bookmarks files in the upper pane on your workstation. This is a client-calculated field.

**Flag User List (Advanced)**

Values: text displayed as a list of user names. For example: `[Greg, Sam]` indicates user names.



	Internal Identifier: <code>FlagUserList</code> .
	Can be used in queries. Identifies users who have set flags on a given item.
<b>Folder Path</b>	Values: <code>text</code> .
	Internal Identifier: <code>Folder Path</code> (contains spaces).
	The path to the folder. This is not the path to the working folder.
<b>Item Deleted By</b>	Values: list of users, <code>None</code> .
	Internal Identifier: <code>ItemDeletedUserID</code> .
	The name of the user who deleted this item. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.
<b>Item Deleted Time</b>	Values: <code>date/time</code> .
	Internal Identifier: <code>ItemDeltedTime</code> .
	The time at which the item was deleted. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.
<b>Locked By</b>	Values: list of users, <code>&lt;None&gt;</code> .
	Internal Identifier: <code>ExclusiveLocker</code> .
	The name of the user who has exclusively locked a folder.
<b>Modified By</b>	Values: list of users, <code>&lt;None&gt;</code> .
	Internal Identifier: <code>ModifiedUserID</code> .
	The name of the user who last modified the item.
<b>Modified Time</b>	Values: <code>date/time</code> .
	Internal Identifier: <code>ModifiedTime</code> .
	The time at which an item was last modified. The item may have been checked in or had its properties changed. For folders, this has nothing to do with the working folder. Use <b>Local Time Stamp</b> for the time a working folder was last modified.
<b>My Lock</b>	Values: <code>Exclusively Locked By Me</code> , <code>Non-exclusively Locked By Me</code> , <code>Not Locked By Me</code> .
	Internal Identifier: <code>MyLock</code> .
	Indicates whether the current user has the item locked and, if so, whether that lock is exclusive or not. This is a client-calculated field.
<b>New Revision Comment (Advanced)</b>	Values: <code>text</code> .
	Internal Identifier: <code>NewRevisionComment</code> .
	Internal use only. The client uses this value during the item update process. The field always appears empty if added to the upper pane. This is a client-calculated field.
<b>Non-Exclusive Lockers</b>	Values: <code>text</code> .
	Internal Identifier: <code>NonExclusiveLockers</code> .
	The names of the users who have locked the folder non-exclusively.
<b>Object ID</b>	Values: <code>number</code> .

	Internal Identifier: <code>ID</code> .
	Each item is assigned an object ID when it is added to a view. For applicable items, when it is branched in a child view, it is assigned another object ID. The original ID belongs to the folder in the parent view.
<b>Parent ID (Advanced)</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>ParentID</code>.</p> <p>The object ID of an item in the parent view. The Parent ID is <code>-1</code> if this view has no parent view.</p>
<b>Priority</b>	<p>Values: <code>High</code>, <code>Low</code>, <code>Normal</code>.</p> <p>Internal Identifier: <code>Priority</code>.</p> <p>The value of the <b>Priority</b> field. You can use repository customization to change the names of these values or include other values.</p>
<b>Read Only (Advanced)</b>	<p>Values: <code>No</code>, <code>Yes</code>.</p> <p>Internal Identifier: <code>ReadOnly</code>.</p> <p>Indicates whether the item's configuration is read-only (as in a rollback configuration of a view)/its behavior does not allow it to branch on modification. For folders, do not confuse a read-only configuration (an application issue) with a read-only folder (an operating system issue). A read-only folder cannot be edited and saved to disk. A folder whose configuration is read-only can be edited and saved to disk; it just cannot be checked in.</p>
<b>Read Status</b>	<p>Values: <code>Read</code>, <code>Unread</code>.</p> <p>Internal Identifier: <code>ReadStatus</code>.</p> <p>Indicates whether an item is considered read or not read. This is a client-calculated field.</p>
<b>Read Status User List</b>	<p>Values: <code>text</code> displayed as a list of user names. For example: <code>[Greg, Sam]</code> indicates user names.</p> <p>Internal Identifier: <code>ReadStatusUserList</code>.</p> <p>Can be used in queries. Identifies users for whom a given item's status is <b>Unread</b>.</p>
<b>Recipient Count</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>RecipientCount</code>.</p> <p>The number of recipients to whom a topic is addressed.</p>
<b>Recipient IDs</b>	<p>Values: <code>byte array</code>, displayed as a bracketed series of numbers in hex format. For example, <code>[14 00 00 00]</code> indicates a specific user.</p> <p>Internal Identifier: <code>RecipientIDs</code>.</p> <p>Can not be used in queries. The ID numbers assigned to the users who are recipients (people to be notified about this topic).</p>
<b>Recipient Names</b>	<p>Values: <code>text</code> containing a series of users names separated by spaces.</p> <p>Internal Identifier: <code>Recipient Names</code>.</p> <p>The names of the recipients designated for notification about this topic.</p>

<b>Revision Flags (Advanced)</b>	<p>Values: 0.</p> <p>Internal Identifier: RevisionFlags.</p> <p>Internal use only.</p>
<b>Share State</b>	<p>Values: DerivedShare, Not Shared, Root Share.</p> <p>Internal Identifier: ShareState</p> <p>Indicates whether this item is shared. <code>Not Shared</code> means that the item is not shared. <code>Root Share</code> means that the item is shared and this item is the original (or root) reference. <code>DerivedShare</code> means that the item is shared, but this item is not the original (or root) reference.</p>
<b>Short Comment</b>	<p>Values: text.</p> <p>Internal Identifier: ShortComment.</p> <p>Stores the initial 2000 characters provided as the reason for changing an item's properties or contents. Additional text is stored in the <b>Comment</b> field.</p>
<b>Status</b>	<p>Values: Active, Inactive.</p> <p>Internal Identifier: Status.</p> <p>Indicates the status of this topic.</p>
<b>Title</b>	<p>Values: text.</p> <p>Internal Identifier: Title.</p> <p>The text of the <b>Title</b> field.</p>
<b>Topic Number</b>	<p>Values: number.</p> <p>Internal Identifier: TopicNumber.</p> <p>The number assigned to a topic. For example, if the <b>Object ID</b> is 0, the topic number is 1.</p>
<b>Type</b>	<p>Values: Response, Topic.</p> <p>Internal Identifier: Type.</p> <p>Indicates whether the item is a topic (root of a topic tree) or a response (branch of a topic tree). This is a client-calculated field.</p>
<b>Version (Advanced)</b>	<p>Values: number.</p> <p>Internal Identifier: RevisionNumber.</p> <p>The last number in the branch revision number. For example, if the branch revision number is 1.3.1.2, the version is 2.</p>

## Topic Properties

This topic presents the topic properties and their descriptions as displayed in the **Topic Properties** dialog box. The **Topic Properties** dialog box contains the following tabs of properties.

### Topic

The following properties are on the **Topic** tab.

<b>Title</b>	Displays the title of the topic.
<b>Created By</b>	Displays the name of person who created the topic.
<b>Created On</b>	Displays the date on which topic was created.
<b>Attachments</b>	Displays the number of attachments to the topic.
<b>Modified By</b>	Displays the name of the last person who modified the topic.
<b>Modified On</b>	Displays the date on which the topic was last modified.
<b>Content</b>	Displays the text contents of the topic.

## Options

The following properties are on the **Options** tab.

**Recipients** Displays the list of intended recipients of the topic or response.

**Note:** You cannot delete yourself as a recipient unless you delete all the recipients. When there are recipients, StarTeam does not allow you to remove yourself from the notification list.

**Priority** Displays the topic priority: **Low**, **Normal**, or **High**

**Status** Displays the topic status: **Active** or **Inactive**.

## Custom

You can create custom properties for an item which will display in the item **Properties** dialog box.

The following properties are on the **Custom** tab.

**Property** Displays each custom property name.

**Value** Displays the values for each custom property. Double-click the property name to edit the value.

## Attachments

The **Attachments** page contains a list of all the files attached to the current topic.

## Comment

The following properties are on the **Comment** tab.

**Comment For This Revision** Displays the reason for the changes to the current revision.

**Comment For New Revision** Displays the reason for the changes to the new revision.

# Files

To place a file under version control, it must be added to a folder in a StarTeam project view, which stores a copy of the file in the StarTeam repository. After the file has been added to StarTeam, you and other members of your team can check it out, revise it, and check in new revisions, while StarTeam maintains information on all revisions of the file. Note that all check-ins in StarTeam are atomic.

When checking out a file revision, you should verify that you have the *tip* or latest version of the file. Doing this ensures that the file you see contains the latest changes. If you intend to modify the file, you should check it out with an exclusive lock, to indicate to others that you are working on it.

When you check a file in, StarTeam records the file changes as a new revision. As part of the check-in process, you can remove the lock, notifying others that the file is available, or maintain the lock, showing

that you intend to continue working on the file. If two team members change the same text file simultaneously or if one member changes an outdated file, StarTeam contains a merge option that allows the file changes to be combined so that no work is lost. In such cases, StarTeam assigns a *Merge* status to the file.



**Note:** The SDK, StarTeam Server, and most clients support files larger than 4 GB. If you plan on taking advantage of large file support, you should upgrade all users to the current StarTeam version. Large file sizes are not compatible with older StarTeam versions.



**Note:** Using the StarTeam Eclipse Plugin, operations for files are surfaced using the:

- **StarTeam** main menu commands.
- Context menus provided in the various StarTeam views.
- StarTeam-specific toolbar buttons.
- **Team** context menu.
- **Team Synchronizing** perspective.

### Files Under Version Control

If a file resides in the working folder of an application folder, you can add that file to the application folder. This operation places that file under version control. A copy of the working file becomes the first revision of that file stored in the repository. If the working file is deleted later, the data is not lost because a copy exists in the repository. The application creates a new revision of this file in the repository every time you check the file in.

Every time you check a file revision out, its contents are copied to a working folder. Checking out a revision also ensures that you have the tip or a specific revision to work on. For example, you may need a team member's most recent changes to a file, or you may have deleted the working file from your hard drive and now need another copy.

The application enables you to label the tip revisions of every item within a view. For example, when the project reaches a particular milestone (such as beta), you might give the view's items a label, called a view label. Then you can configure the view to return to the way it was at the time the label was applied, check out revisions as a group using that label, create a new view based on the label, or assign the label to a promotion state.

The application also provides revision or version labels. You can label one or more revisions as you check them in or by applying the label to each of the revisions using the **Labels** command on the File menu. StarTeam makes it easy to check out those files as a group using the label. A file revision can have any number of labels. However, no two revisions of the same file in the same view can have the same label.

### Recommendations for Working with Files Under Version Control

Here are some recommendations about using files under version control:

- To let other team members know that you intend to make changes to a file, change the lock status to *exclusive* as part of the check-out procedure.
- As part of the check-in process, you can notify others both that you are finished making your changes to the file and that it is available for them to check out by removing the lock status.
- If you intend to continue making changes to the file but still want to check it in for backup purposes, keep the file locked.
- If two team members change the same text file simultaneously or if one member changes an outdated file, you can use the merge option to combine the changes in these files so no work is lost. In such cases, the application gives the file a *Merge* status.
- To prevent yourself from changing a file that you have not locked, select **Window > Preferences > Mark Unlocked Working Files Read-only** the **Mark Unlocked Working Files Read-only** preference. Then, if you check out a file that you have not locked, the working copy becomes read-only.

## Opening Files

1. Select the file in the Server Explorer.
2. Right click the file and choose **Open**.

If the file does not open in an associated application, an association may not have been created for the selected file type. You can set file associations in Eclipse using the **Preferences** dialog box (**Window > Preferences**).

## Modifying File Properties

1. Open a StarTeam Perspective, and select a file from the Server Explorer or in one of the Eclipse Explorers.
2. Lock the file.
3. Right click and select **Properties**. The **Properties** dialog box opens.



**Note:** If you open the **Properties** dialog box from one of the Eclipse Explorers, expand **StarTeam Provider** and select **File** from the tree-view list.

4. On the **General** tab, you can change the file name and description or set the **Executable** flag. Changing a file name also changes name of the working file.
5. Select the **Archive** tab where you can optionally change the following settings:
  - Change the **Compression** setting to **None**, **Maximize speed**, **Default** (a compromise between speed and compression), or **Maximize compression**.
  - Select the **Store versions as deltas** check box to store text files using the forward delta method. Clear it to store file revisions in entirety.
6. Select the **Custom** tab.

If your company uses StarTeam Repository Customization feature, your administrator may have created additional file properties.

  - a) To change a custom property, double-click on its name. The **Edit Property** dialog box opens.
  - b) Select a new value for the property.

For integer, text, and real fields, **Value** is a text box. For enumerated types and user IDs, it is a list box. For dates and times, **Value** has a date check box and a time check box, each of which is followed by a date or time in the format for your locale.
7. Select the **Comments** tab.
8. Type the reason for changing the file's properties in the text box.

## Enabling Concurrent File Editing

The personal option named **Use Non-Exclusive Locks In Integrations** affects how files are locked when accessed from application integrations such as Visual Studio. If you select this option, locking a file (for example, as part of a check-out operation) creates a non-exclusive lock rather than an exclusive lock.

With an exclusive lock, only the person who has the file locked can check in the file. With a non-exclusive lock, others can check in the file. Exclusive locks are the safest, but non-exclusive locks are often preferred because text files can be easily merged using File Compare/Merge. Using non-exclusive locks allows more than one person to edit a file at one time. If team members are not editing the same lines of the file, the merged file usually has no conflicts.

If you are using an application integration for your development environment, for example, the integration with Visual Studio, you cannot check in files from the development environment if both the **Require Exclusive Lock When Files Are Checked In** check box in the **Project Properties** dialog box, (**Options** tab) and the **Use Non-exclusive Locks In Integrations** check box in the **Personal Options** dialog box

(**Files** tab) are checked. The administrator usually determines the setting of the **Require Exclusive Lock When Files Are Checked In** check box. However, personal options are set by you for your workstation.

1. Choose **Window > Preferences**. The **Preferences** dialog box opens.
2. Expand **Team > StarTeam**.
3. Click **Files**.
4. Check **Use Non-exclusive Locks In Integrations**.
5. Click **OK**.



**Note:** If you have checked **Use Non-exclusive Locks In Integrations** and experience check-in problems, try clearing the check box. You may want to talk to your administrator about the setting for the **Require Exclusive Lock When Files Are Checked In** check box.

## Excluding Files from a Project

Some types of files will never be added to a project, although they may reside in a working folder. For example, suppose you are creating files with an application that makes an automatic backup (.bak) copy of each file every time you save the file. Although your working folder might contain several .bak files, you would have no reason to check them into (or out of) the application. Therefore, you should exclude them from the project view.

Exclude lists can also be inherited from parent folders.

1. With a StarTeam perspective or view open, choose **StarTeam > Folder > Properties**.
2. Select **Folder**.
3. Select the **Exclude** tab.



**Note:** The **Exclude** tab does not affect files that are already part of the project.

4. Do one of the following:

<b>Inherit and Use Local Exclude List</b>	Excludes files that match the exclude list specifications set for this folder as well as those of its parent folder. If the <b>Local Exclude List</b> text box does not yet include any file specifications, add them.
<b>Use Local Exclude List</b>	Excludes files that match the exclude list specifications set for this folder. If the <b>Local Exclude List</b> text box does not yet include any file specifications, add them.
<b>No Exclude List</b>	Includes all files.

5. Type one or more file specifications to use for matching files.

Use standard expressions (with \* and ? wild cards) separated by commas, spaces, or semicolons. To include a comma, space or semicolon as part of the specification, enclose the specification in double quotes.

A trailing / character means that **Not-in-View** folders will be excluded. For example, bin/ would cause all **Not-in-View** folders named bin to be excluded from the folder tree.



**Note:** The \ character does not work. It is treated as an escape character.

## Finding Files Associated with Active Process Items

When you have files associated with an active process item, you can quickly find all associated file changes by following these steps.

1. Open the view that contains the active process item.

You can see what item is the active process item by looking at the left side of the **Status Bar**. The second box in the **Status Bar** displays the **Active Process Item** icon, followed by the name of the item.

2. Open the **Link** view.
3. Click **Link with Selection** in the toolbar.
4. Select the active process item in the appropriate Change Request, Requirement, or Task view.
5. Right click on a link and choose **Select Linked Item**. The view opens containing the target of the associated item.

## Marking Unlocked Files Read-only

In many cases, users make edits before realizing that their files must be exclusively or non-exclusively locked to check them in. If the files are read-only, users are less likely to make this mistake.

1. Choose **StarTeam > Project > Properties**. In the Server, Navigator, or Package Explorers, right click on the project, and choose **Properties** from the context menu..

The **Properties** dialog box opens.

2. Select **Project**.
3. Select the **Options** tab.
4. Check **Mark Unlocked Working Files Read-only**, which applies to files that are unlocked in the application or in application integrations with third-party applications.

If this check box is cleared, you must use the operating system to change the read-only attribute to read/write.

Working copies of unlocked files will now become read-only when the following file operations are performed:

- File check-ins.
- File check-outs (from the **File** or **History** pane).
- File unlocks.



**Note:** This project property overrides the identical **Mark Unlocked Working Files Read-only** personal option. If you change your mind after selecting the property (or the equivalent personal option), verify that no files are writable before clearing the check box. Next, force a check out and lock all the files (or just the read-only files). Finally, unlock them.

## Setting the File Executable Bit for UNIX

When you add a file from a UNIX operating system, the state of the executable bit is preserved by StarTeam. For each file, there is an **Executable** check box that becomes selected if the executable bit is set and becomes cleared if the bit is not set. Future check-out operations ensure that the executable bit for the checked-out file matches the setting of the **Executable** check box.

1. Select a file in the Server Explorer or any of the Eclipse Explorers.
2. Right-click on the file, and choose **Properties**. The **Properties** dialog box opens.
3. On the **General** tab, check or uncheck **Executable**.
4. Click **OK**.

## Viewing Annotations

1. Select a text file in the Server Explorer, Navigator, or Package Explorer.
2. Choose **Window > Show View > Annotate**.

*Alternative:* Right-click the selected file and choose **Team > Show Annotations**.



The **Annotate** view displays the contents of the selected file, and the annotation information in the left gutter of the view.

3. Click **Info** in the gutter to view the details about a specific revision.

## Viewing Previous File Revisions

You can review the contents of a prior file revision in either the default editor or in the application for which the file type is registered.


1. Open the History view, and click **Link with Selection**.
2. Select a file in the Server Explorer or from one of the Eclipse Explorers..
3. In the **History** view, select the specific revision you want to review.
4. Right-click the selected item to open the context menu and choose one of the following:

<b>View Revision Content</b>	Copies the revision to a temporary file and display it in the default editor (Notepad or the alternate editor specified in the <b>Personal Options</b>
<b>Open Revision Content</b>	Copies the revision to a temporary file and display it in the associated application.



**Note:** The client creates the temporary files in the local temp directory on the system. For example, if working on a Microsoft Windows system, the temporary files are created in the C:\Documents and Settings\\Local Settings\Temp directory. When you exit the client, the files are deleted from the system.

## Comparing and Merging Files

 **Important:** You must specify File Compare/Merge as the alternate compare/merge tool before you can use it to compare and merge files.

This procedure presents the basic high-level task involved in comparing and merging a local file with a file in the StarTeam repository, which opens the File Compare/Merge component.

1. There are two ways to launch File Compare/Merge. Do one of the following:
  - In the **History** view, select two files to compare.
  - In the Package Explorer view, select a StarTeam repository file in the view to compare with your local file or select two files to compare.
2. Done one of the following from the context menu:
  - Right-click and select **Merge File Contents**.
  - Right-click and select **Compare With > Remote File in External Comparison Application** or **Compare With > Each Other in External Comparison Application**.

The File Compare/Merge session opens in a separate view or window, with each file in its own edit pane.

3. Perform any required edits to the files directly in the edit panes.



**Tip:** You can find additional procedures for Comparing and Merging Files in the online help of the File Compare/Merge tool.

## Converting Resource Change Status

Converting the change status does not require check in or check out as it only affects local sync data, and it is reversible.

1. Right-click the file or folder in the **Navigator** view. and choose Properties.

*Alternative:* Right-click the file or folder in the **Synchronize** view and choose **Convert To**.

2. Choose **Properties**.
3. Click the appropriate button for the desired change on the **StarTeam Provider** page.
4. Click **OK**.

## Editing Check-in Comments

1. Select the file or multi-select files in one of the Eclipse Explorers.
2. Right-click, and choose **Team > Comment for Next Check-in > Edit**. The **Comment** dialog box opens.



**Note:** To remove a previously-added check-in comment, choose **Team > Comment for Next Check-in > Remove**.

3. Type the text you want for the comment.



**Tip:** Comment fields allow up to 30,000 characters.

4. Click **OK**.



**Note:** You can also open the **Check In** dialog box and use the toolbar in the **Changes** area to edit and remove check-in comments.

## Enabling Quick Diff in Text Editors

1. Choose **Window > Preferences** The **Preferences** dialog box opens.
2. Expand the **General > Editors > Text Editors** nodes.
3. Select **Quick Diff**.
4. Check **Enable Quick Diff**.
5. Under the option, **Use this reference source**, choose **Latest StarTeam Revision** from the list box.
6. Click **OK**.

After activating Quick Diff and opening a text editor, the text editor displays colored gutter bars on the left to indicate differences from the latest StarTeam revision of the opened file. Hover text displays the differences as you hover your mouse over the gutter bars.

## Force Check-out with Keyword Expansion

This process allows you to get the local file contents replaced by the remote version and thus have keywords expanded after the project settings for keywords were changed.

1. Select the workspace file, and choose **Replace with > Revision** from the context menu.
2. Select the latest revision. Doing so does not change the file; however any keywords in the file (such as, \$Header\$) are expanded.

## Locking and Unlocking Items

Before changing the contents of a file or editing item properties, you should exclusively lock the file or item. This action informs other team members that you intend to make changes.

1. Select one or more items.
2. Click the **Lock** or **Unlock** button on the toolbar.

The selected items become exclusively locked, and you are listed as the user who has locked them.

To lock an item using a context menu

1. Open the appropriate view to select a folder, file, change request, task, topic, or requirement.
2. Right-click an item, and choose **Lock/Unlock**. The **Set My Lock Status** dialog box opens.



**Note:** If setting a lock for a file in the Navigator or Package Explorers, choose **Team > Lock/Unlock**.

3. Select a lock status option:
  - **Unlocked:** Removes your exclusive or non-exclusive lock on the selected items.
  - **Exclusive:** Prevents others from creating new revision of this item (until you release the lock or another person breaks your lock).
  - **Non-exclusive:** Indicates that you are working on the item and may possibly make changes (not recommended for items other than files).
4. *Optional:* Check **Break Existing Lock** to break another team member's lock on the item. If e-mail is enabled, StarTeam will send an e-mail message to the team member whose lock has been broken to inform him or her of this fact.



**Note:** You must be granted the appropriate privileges to be able to break another person's locks.

## Modifying the EOL Conversion Mode

StarTeam Eclipse uses MD5 hash values for status calculation and revision matching. Therefore, it needs to know what EOL conversions were used for a file at the last file check-in and check-out operation. StarTeam Eclipse needs to remember these EOL conversions to adjust the locally computed MD5 hash value to the remotely reported ones. For instance, when using the Cross-Platform Client and StarTeam Eclipse in parallel, operations performed by the Cross-Platform Client bypass the StarTeam Eclipse EOL persistency, and result in incorrectly reported statuses.

1. Right-click the file in the **Navigator** view and choose **Properties**.
2. Select the desired EOL conversion mode on **StarTeam Provider** page: **Windows**, **Unix**, **Mac**, or **None**.
3. Click **Apply** or **OK**.

## Viewing or Modifying Item Properties

This section explains how to use the standard properties dialog to edit item properties. Depending on how your team has set up the application, you may see a totally different dialog box called an alternate property editor (APE).

Every time the properties of an item are modified, a new revision of that item is created. If you modify a property, you should also create a revision comment explaining the modification using the **Comment** tab provided in the **Properties** dialog box.

1. Select an item in one of the StarTeam or Eclipse views.
2. Do one of the following:
  - Select an item and double click on it.
  - Right-click the folder or item and choose **Properties**.
3. Modify any of the property fields in the corresponding **Properties** dialog box that opens, then click **OK**.

## Sample File Detail View Template

You can use the following sample for customizing the **Detail** view for the File component.

```
<html>
<head></head>
<body>
<table bgcolor=#aaabbbccc width=100%>
<tr>
<td align=center><b>~~Name~~<b></td>
```

```

</tr>
</table>
<table>
<tr>
<td align=left><b>Status:</b></td>
<td>~~Status~~</td>
</tr>
<tr>
<td align=left><b>Size:</b></td>
<td>~~FileSize~~</td>
</tr>
<tr>
<td align=left><b>Working folder:</b></td>
<td>~~Path~~</td>
</tr>
<tr>
<td align=left><b>Project Folder Path:</b></td>
<td>~~Folder Path~~</td>
</tr>
</table>
<hr>
<b>Last modified by: </b>~~Author~~, ~~Date&Time~~<br>
<b>Comment:</b><i>~~Comment~~</i>
</body>
</html>

```

### Fields Used in Detail View Templates

The fields used in the **Detail** view HTML templates are recognized by the client when they are contained between double tilde `~~` characters. For example: `~~Path~~` represents the path to your local working folder represented by the **Path** field found in the **Working File** tab of the **File Properties** dialog box.

## File Properties

This topic contains the file properties and their descriptions as displayed in the **File Properties** dialog box. The **File Properties** dialog box contains the following tabbed pages of properties.

### General

The following properties are on the **General** tab.

<b>Name</b>	Displays the name of the file.
<b>Description</b>	Displays the description of the file.
<b>Status</b>	Indicates the relationship between the copy of the file in your working folder and the tip revision in the repository.
<b>Size</b>	Displays the size of the tip revision of the file in bytes.
<b>Last Modified By</b>	Displays the name of the person who last modified the file.
<b>Last Modified On</b>	Displays the date on which the file was last modified.
<b>File Time Stamp</b>	Displays the time at which the file was last modified.
<b>Locked Exclusively By</b>	Displays the name of the user who has exclusively locked the file.
<b>Locked Non-Exclusively By</b>	The name of user who has non-exclusively locked a file.
<b>EOL check-out format</b>	Displays the options for EOL formatting for text files during check-out:

**Client Defined:** Causes workstation default or per-checkout EOL conversion option to be used.

**Fixed CR, Fixed LF, and Fixed CRLF:** Causes this EOL format to be used always; the workstation/check-out conversion option is ignored.

**File type** Indicates which file type is selected: ASCII, Binary, or Unicode.  
**Executable** Indicates whether the executable bit should be set for a UNIX file.

### Working File

The following properties are on the **Working File** tab.

**Path** Displays the Path to the working folder for the file.  
**File Exists** Indicates whether the file exists in the working folder.  
**Size** Displays size of the file in the working folder in bytes.  
**Time Stamp** Displays the time that the working folder file was last modified.  
**Executable** Indicates whether the executable bit should be set for a UNIX file.

### Custom

You can create custom properties for an item which will display in the item **Properties** dialog box.

The following properties are on the **Custom** tab.

**Property** Displays each custom property name.  
**Value** Displays the values for each custom property. Double-click the property name to edit the value.

### Comment

The following properties are on the **Comment** tab.

**Comment For This Revision** Displays the reason for the changes to the current revision.  
**Comment For New Revision** Displays the reason for the changes to the new revision.

## File Fields

This section lists all the file fields in alphabetical order.



**Note:** Client-calculated fields cannot be used in custom email notifications or StarTeam Notification Agent. Reports can use any field name.

**Archive Format (Advanced)** Values: `Native-II`.  
Internal Identifier: `ArchiveFormat`.  
Indicates whether the format in which file revisions are stored. StarTeam 2006 uses only `Native-II` format.

**Archive Path** Values: `text`.  
Internal Identifier: `ArchivePath`.  
The path to the PVCS archive or VSS project containing a file.

**Archive Name (Advanced)** Values: 32-digit hex string representing the MD5 value of the file.

Internal Identifier: `STArchiveName`.

For a file stored in a Native-II Vault, indicates the name of the file that stores the tip revision. This name is the MD5 value of that file revision's content, converted to a 32-digit hex string.

### Archive Type

Values: `Native`, `PVCS`, `VSS`.

Internal Identifier: `Type`.

Indicates whether a file is stored as a StarTeam (Native), PVCS, or VSS file.

### Branch On Change (Advanced)

Values: `No`, `Yes`.

Internal Identifier: `BranchOnChange`.

Indicates whether the item will branch when it changes.

The value is `No` if the item's behavior is not set to **Branch On Change**. Reasons for this may be:

- The item is in the root or a reference view and the **Branch On Change** feature is disabled.
- The item is in a branching view but has already branched as a result of a change, which, in turn, results in the **Branch On Change** feature becoming disabled.
- The item is in a branching view, but its behavior currently does not permit it to branch on change. This means that modifications are checked into the parent view.



**Note:** If the value is `No`, the value of the **Branch State** explains the `No`.

### Branch State (Advanced)

Values: `Branched`, `Not Branched`, `Root`.

Internal Identifier: `BranchState`.

Indicates whether an item has branched in the child view, is still unbranched (and therefore is part of the parent view), or was created in the view in which it resides.

The values `Branched` and `Not Branched` apply to items in branching views. The value `Root` applies to items created in the view in which the item currently resides.

If the view is a reference view, it reflects the state of the item in the reference view's parent.

### Comment

Values: `text`.

Internal Identifier: `Comment`.

The initial 2000 characters provided as the reason for changing an item's properties or contents are stored in the **Short Comment** field. The **Comment** field stores those 2000 characters and any additional text. Changing an item's properties causes the application to create a new revision.



**Note:** To include a **Link** comment, the **Comment** field is the value to use in an HTML report.

### CommentID (Advanced)

Values: `number`.

Internal Identifier: `CommentID`.

The ID number assigned to the revision comment. Displays `-1` if no revision comment was supplied.

**Compression Level**

Values: Default, Maximize Compression, Maximize Speed, None.

Internal Identifier: `Compression`.

Indicates a file's level of compression.

<b>Default</b>	A compromise between <b>Maximize Compression</b> and <b>Maximize Speed</b> .
<b>Maximize Compression</b>	The densest possible compression of file revisions. Used to save space on the server.
<b>Maximize Speed</b>	The fastest possible compression of file revisions. Used to improve server performance.
<b>None</b>	No compression.

**Configuration Time**

Values: date/time.

Internal Identifier: `ConfigurationTime`.

Indicates the time to which an item is configured. If you configure an item to a specific time, this field contains that time. If you configure an item to a label or promotion state, this field shows either the time at which the label was created, or the time at which the label associated with the promotion state was created.

**Content Revision**

Values: number.

Internal Identifier: `ContentVersion`.

The number of times a file has been checked in. If the file is in a child view, it includes all the content revisions from the parent view in this number. Each revision appears in the file's history.

**Created By**

Values: list of users, <None>.

Internal Identifier: `CreatedUserID`.

The name of the user who created the first revision in the view. This is either the user who added the item to the project, or the user who checked in the revision that branched.

**Created Time**

Values: date/time.

Internal Identifier: `CreatedTime`

The time at which the first revision in the view was created.

**Deleted By**

Values: list of users, <None>.

Internal Identifier: `DeletedUserID`.

The name of the user who deleted the item. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.

**Deleted Time**

Values: date/time.

Internal Identifier: `DeletedTime`.

The time at which an item was deleted. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.

**Description**

Values: text.

Internal Identifier: `Description`.

	The description provided for an item at the time it was added to the view, including any later edits to it.
<b>Dot Notation</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>DotNotation</code>.</p> <p>The branch revision number, for example, <code>1.2.1.0</code>.</p>
<b>End Modified Time (Advanced)</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>EndModifiedTime</code>.</p> <p>The date and time at which a revision ceased to be the tip revision. Although this field can be displayed in the upper pane, its value is always blank. This is because, at any given configuration time, the item is still the tip revision.</p>
<b>EOL Character</b>	<p>Values: numeric representation of ANSI character.</p> <p>Internal Identifier: <code>EOL</code>.</p> <p>For internal use only. This field is primarily used to determine an ANSI character to use when breaking up lines within files for delta storage with Native-1 Vaults.</p>
<b>Executable</b>	<p>Values: <code>No</code>, <code>Yes</code>.</p> <p>Internal Identifier: <code>Executable</code>.</p> <p>Indicates whether the executable bit should be set for a UNIX file.</p>
<b>Extension</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>Extension</code>.</p> <p>Displays the extension of the file. This field is client-calculated.</p>
<b>File Time Stamp at Check-In</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>Modified</code>.</p> <p>The file's time stamp at the time it was last checked in.</p>
<b>File Type</b>	<p>Values: <code>ASCII</code>, <code>Binary</code>, <code>Unicode</code>.</p> <p>Internal Identifier: <code>Charset</code>.</p> <p>Indicates whether the file is an ASCII (text), binary, or Unicode.</p>
<b>Flag</b>	<p>Values: <code>No</code>, <code>Yes</code>.</p> <p>Internal Identifier: <code>Flag</code>.</p> <p>Marks or bookmarks files in the upper pane on your workstation. This is a client-calculated field.</p>
<b>Flag User List (Advanced)</b>	<p>Values: text displayed as a list of user names. For example: <code>[Greg, Sam]</code> indicates user names.</p> <p>Internal Identifier: <code>FlagUserList</code>.</p> <p>Can be used in queries. Identifies users who have set flags on a given item.</p>
<b>Folder</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>Folder</code>.</p> <p>The name of the folder that stores the item. This is a client-calculated field.</p>



<b>Folder Path</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>Folder Path</code> (contains spaces).</p> <p>The path to the folder. This is not the path to the working folder.</p>
<b>Hive ID (Advanced)</b>	<p>Values: number assigned by the StarTeam Server.</p> <p>Internal Identifier: <code>HiveID</code>.</p> <p>Indicates the ID number of the hive that stores the tip revision for a file stored in a Native-II Vault.</p>
<b>Item Deleted By</b>	<p>Values: list of users, <code>None</code>.</p> <p>Internal Identifier: <code>ItemDeletedUserID</code>.</p> <p>The name of the user who deleted this item. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.</p>
<b>Item Deleted Time</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>ItemDeltedTime</code>.</p> <p>The time at which the item was deleted. Because deleted items do not appear in the list, this information is unavailable to users. Internal Use Only.</p>
<b>Local Name</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>LocalName</code>.</p> <p>Name of the working file. This is a client-calculated field.</p>
<b>Local Path</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>LocalPath</code>.</p> <p>Path name to the folder/file. This is a client-calculated field.</p>
<b>Locked By</b>	<p>Values: list of users, <code>&lt;None&gt;</code>.</p> <p>Internal Identifier: <code>ExclusiveLocker</code>.</p> <p>The name of the user who has exclusively locked a folder.</p>
<b>MD5 Checksum</b>	<p>Values: <code>byte array</code>. Displayed as a bracketed series of numbers in hex format. The StarTeam client displays only significant zeroes so the <code>05</code> and <code>0A</code> would become just <code>5</code> and <code>A</code>, and <code>A-F</code> as <code>a-f</code>.</p> <p>Internal Identifier: <code>MD5</code>.</p> <p>Cannot be used in queries. The MD5 checksum for the tip revision.</p>
<b>Modified By</b>	<p>Values: list of users, <code>&lt;None&gt;</code>.</p> <p>Internal Identifier: <code>ModifiedUserID</code>.</p> <p>The name of the user who last modified the item.</p>
<b>Modified Time</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>ModifiedTime</code>.</p> <p>The time at which an item was last modified. The item may have been checked in or had its properties changed. For folders, this has nothing to do with the working folder. Use <b>Local Time Stamp</b> for the time a working folder was last modified.</p>

<b>My Lock</b>	<p>Values: Exclusively Locked By Me, Non-exclusively Locked By Me, Not Locked By Me.</p> <p>Internal Identifier: MyLock.</p> <p>Indicates whether the current user has the item locked and, if so, whether that lock is exclusive or not. This is a client-calculated field.</p>
<b>Name</b>	<p>Values: text.</p> <p>Internal Identifier: Name.</p> <p>Displays the name of the item.</p>
<b>New Revision Comment (Advanced)</b>	<p>Values: text.</p> <p>Internal Identifier: NewRevisionComment.</p> <p>Internal use only. The client uses this value during the item update process. The field always appears empty if added to the upper pane. This is a client-calculated field.</p>
<b>Non-Exclusive Lockers</b>	<p>Values: text.</p> <p>Internal Identifier: NonExclusiveLockers.</p> <p>The names of the users who have locked the folder non-exclusively.</p>
<b>Object ID</b>	<p>Values: number.</p> <p>Internal Identifier: ID.</p> <p>Each item is assigned an object ID when it is added to a view. For applicable items, when it is branched in a child view, it is assigned another object ID. The original ID belongs to the folder in the parent view.</p>
<b>Parent Branch Revision (Advanced)</b>	<p>Values: number.</p> <p>Internal Identifier: ParentRevision.</p> <p>The last digit in the branch revision number before an item branched. For example, if this number is 7, the branch revision was 1.7 at the time the item branched (becoming 1.7.1.0, as seen in the item's history). This number is -1 if an item was not inherited from the parent view.</p>
<b>Parent ID (Advanced)</b>	<p>Values: number.</p> <p>Internal Identifier: ParentID.</p> <p>The object ID of an item in the parent view. The Parent ID is -1 if this view has no parent view.</p>
<b>Parent Revision (Advanced)</b>	<p>Values: number.</p> <p>Internal Identifier: PathRevision.</p> <p>The revision number at which an item branched. For example, if this number is 8, this item's revision number in the parent view was 8 at the time the item branched. The history should show that revision 9 in the first revision in the current view. This number is 0 if this item was not inherited from the parent view.</p>
<b>Path</b>	<p>Values: text.</p> <p>Internal Identifier: Path</p> <p>The path to an item's working folder. This is a client-calculated field.</p>

<b>Project ID (Advanced)</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>ProjectID</code>.</p> <p>The ID number assigned to a project. Within a server configuration, projects are assigned ID numbers in the order in which they are created. The first project has ID 0.</p>
<b>PVCS Revision (Advance)</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>PVCSRev</code>.</p> <p>The file's revision number in PVCS's dot notation.</p>
<b>Read Only (Advanced)</b>	<p>Values: <code>No</code>, <code>Yes</code>.</p> <p>Internal Identifier: <code>ReadOnly</code>.</p> <p>Indicates whether the item's configuration is read-only (as in a rollback configuration of a view)/its behavior does not allow it to branch on modification. For folders, do not confuse a read-only configuration (an application issue) with a read-only folder (an operating system issue). A read-only folder cannot be edited and saved to disk. A folder whose configuration is read-only can be edited and saved to disk; it just cannot be checked in.</p>
<b>Revision</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>ViewVersion</code>.</p> <p>The number of times a file has been checked in or had its properties changed. If the file is in a child view, it includes all the revisions from the parent view in this number. This is a client-calculated field.</p>
<b>Revision Flags (Advanced)</b>	<p>Values: <code>0</code>.</p> <p>Internal Identifier: <code>RevisionFlags</code>.</p> <p>Internal use only.</p>
<b>Revision on Disk</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>SyncPathVersion</code>.</p> <p>The number of the revision that is currently in the working folder on your workstation. The application displays no number if the file's status is <b>Missing</b>. This is a client-calculated field.</p>
<b>Share State</b>	<p>Values: <code>DerivedShare</code>, <code>Not Shared</code>, <code>Root Share</code>.</p> <p>Internal Identifier: <code>ShareState</code></p> <p>Indicates whether this item is shared. <code>Not Shared</code> means that the item is not shared. <code>Root Share</code> means that the item is shared and this item is the original (or root) reference. <code>DerivedShare</code> means that the item is shared, but this item is not the original (or root) reference.</p>
<b>Short Comment</b>	<p>Values: <code>text</code>.</p> <p>Internal Identifier: <code>ShortComment</code>.</p> <p>Stores the initial 2000 characters provided as the reason for changing an item's properties or contents. Additional text is stored in the <b>Comment</b> field.</p>
<b>Size</b>	<p>Values: <code>number</code>.</p>

	Internal Identifier: <code>FileSize</code> .
	The tip revision's size in bytes.
<b>Status</b>	<p>Values: <code>Current</code>, <code>Deleted on Disk</code>, <code>Deleted on Server</code>, <code>Merge</code>, <code>Missing</code>, <code>Modified</code>, <code>Not In View</code>, <code>Out Of Date</code>, <code>Unknown</code>.</p> <p>Internal Identifier: <code>Status</code>.</p> <p>Indicates the relationship between the copy of an item in your working folder and the tip revision in the repository.</p>
<b>Storage Type (Advanced)</b>	Obsolete.
<b>Sync Branch Version</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>SyncObjectVersion</code>.</p> <p>A field used to determine status. The last number of the branch revision that was most recently checked out to the working folder. This is a client-calculated field.</p>
<b>Sync Content Version</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>SyncContentVersion</code>.</p> <p>A field used to determine status. The revision checked out as the working file or, if the file needs to be merged, a number higher than that. This is a client-calculated field.</p>
<b>Sync Known</b>	<p>Values: <code>No</code>, <code>Yes</code>.</p> <p>Internal Identifier: <code>SyncKnown</code>.</p> <p>A field used to determine status. Indicates whether the server knows the working file's relationship to the tip revision. This is a client-calculated field.</p>
<b>Sync Local Size</b>	<p>Values: <code>number</code>.</p> <p>Internal Identifier: <code>SyncSize</code>.</p> <p>A field used to determine status. The size of the working file in bytes. This is a client-calculated field.</p>
<b>Sync Local Time Stamp</b>	<p>Values: <code>date/time</code>.</p> <p>Internal Identifier: <code>SyncTime</code>.</p> <p>A field used to determine status. The time stamp for the working file. This is a client-calculated field.</p>
<b>Sync MD5</b>	<p>Values: <code>byte array</code>. Displayed as a bracketed series of numbers in hex format. The StarTeam client displays only significant zeroes, so <code>08</code>, <code>0B</code>, and <code>06</code> would become just <code>8</code>, <code>B</code>, and <code>6</code>, and <code>A-F</code> as <code>a-f</code>.</p> <p>Internal Identifier: <code>SyncMD5</code>.</p> <p>Can not be used in queries. A field used to determine status. The MD5 checksum of the working file. This is a client-calculated field.</p>
<b>Sync On Path To Root</b>	<p>Values: <code>No</code>, <code>Yes</code>.</p> <p>Internal Identifier: <code>SyncOnPathToRoot</code>.</p> <p>A field used to determine status. When the working file is not based on the tip revision, this field indicates whether the server knows the relationship between the</p>

two. A `Yes` value in this field means that the working file needs to be merged or is out of date. A `No` value means that the relationship cannot be determined. This is a client-calculated field.

**Vault Branch Version (Advanced)**

Values: `number`.

Internal Identifier: `VaultVersion`.

The number of times a file has been checked in from the current view.

**Version (Advanced)**

Values: `number`.

Internal Identifier: `RevisionNumber`.

The last number in the branch revision number. For example, if the branch revision number is `1.3.1.2`, the version is `2`.

**View**

Values: list of views, `<None>`.

Internal Identifier: `ViewID`.

The name of the view in which the item last branched. For example, if an item is inherited from a parent view but is branched in a child view, the value of this field in the child view changes from the name of the parent view to the name of the child view for the revision that branched and subsequent revisions in the child view.

**Working File Executable**

Values: `text`.

Internal Identifier: `No, Yes`.

Indicates whether the working file is executable. This is a client-calculated field.

**Working File Exists**

Values: `No, Yes`.

Internal Identifier: `LocalFileExists`.

Indicates whether a copy of a file is in its working folder. This is a client-calculated field.

**Working File Size**

Values: `number`.

Internal Identifier: `LocalSize`.

The size of the working file. This is a client-calculated field.

**Working File Time Stamp**

Values: `date/time`.

Internal Identifier: `LocalTimestamp`.

The time stamp of the working file. This is a client-calculated field.

## Check-in and Check-out Operations

One of the main functions of a source control management application is to place files into a project under version control. After it is under version control, team members can check files out, revise them, and check in new revisions. The application preserves historical information about each file revision. And because of its linking capabilities, you can link a file revision to other items that affected the file or a particular revision of it. A number of other operations can be performed on files, such as moving a file or changing its branching behavior. This section specifically addresses the check-in and check-out operations.

## Atomic Check-ins

All check-ins in StarTeam are atomic. Whenever more than one file is checked in as the result of a single transaction all of the files, and their associated process items, are updated in a single action. If for some reason, the check-in fails, none of the files are checked in, and the status of the associated process items is not updated.

For example, suppose `User A` selects to check in all modified files in a StarTeam folder, but one of the selected files is locked by `User B`. Because of the locked file, none of the files are checked in (and none of the process items are updated as fixed) and `User A` is notified that none of the files were checked in because one of the files was locked by `User B`.

## Check-in and Check-out Overview

To place a file under version control, it must be added to a folder in a project view, which stores a copy of the file in the repository. After the file has been added to the repository, you and other members of your team can check it out, revise it, and check in new revisions, while the application maintains information on all revisions of the file. When checking out a file revision, you should verify that you have the tip or latest version of the file. Doing this ensures that the file you see contains the latest changes.

When you check in a file from a working folder on your computer, the application stores it by the MD5 value of its contents. If the file is identified as one that compresses well, it is compressed and placed in the hive's archive with a `.gz` extension. Otherwise, the uncompressed version is placed in the hive's archive.



**Note:** Workspace change packages are created upon the initial check-in of a file. Details about the workspace change package are displayed in the **Change** tab.

When you check out a file, the application copies the requested revision of that file to the appropriate working folder. If a copy of that file is already in the working folder, it is overwritten unless the working file appears to be more recent. In that case, you are prompted to confirm the check out.

You can perform check-out operations on more than one file at a time. For example, you can select files across multiple child folders using the All Descendants button, or you can check out all the files in the selected folder and its descendant folders using the **Check Out All** item on the **File** or context menu. This selection is equivalent to selecting **All Descendants**, all the files in the upper pane, and **Check Out**. When you use the **Check Out All** command, a confirmation dialog appears, regardless of your personal options settings.

### Check-in and Check-out Recommendations

Every time you check a file revision out, its contents are copied to a working folder. Checking out a revision also ensures that you have the tip or a specific revision to work on. For example, you may need a team member's most recent changes to a file, or you may have deleted the working file from your hard drive and now need another copy.

Below are some recommendations about using files that are under version control:

- To let other team members know that you intend to make changes to a file, change the lock status to exclusive as part of the check-out process.
- As part of the check-in process, you can notify others both that you are finished making your changes to the file and that it is available for them to check out by removing the lock status.
- If you intend to continue making changes to the file but still want to check it in for backup purposes, keep the file locked.
- If two team members change the same text file simultaneously or if one member changes an outdated file, you can use the merge option to combine the changes in these files so no work is lost. In such cases, the application gives the file a Merge status.

- To prevent yourself from changing a file that you have not locked, select the **Mark Unlocked Working Files Read-only** personal option. Then, if you check out a file that you have not locked, the working copy becomes read-only.

## Checking In Files

You can check in files:

- Without opening a dialog box to set any specific check in options using the StarTeam toolbar button, **Put outgoing changes to StarTeam Server**.
- By opening a **Check In** dialog box and setting specific check-in options, such as Lock Status, using the Team > Check In command.




**Note:** Unlike other StarTeam clients, the StarTeam Eclipse Plugin does not have a special **Add Files** dialog for putting new files (files with the status in not-in-view) under version control. You use the same standard **Check In** dialog box for files that have been modified while under version control and for files that are new.

1. Choose **Team > Check In** from one of the Eclipse Explorers or from any view that displays the **Team** context menu. The **Check In** dialog box opens.
2. Type a generic reason for the check-in comment, or check **Prompt for a comment (check-in reason) for each file** to open a separate **File Description** dialog box for each file.
3. Select a **Lock status** option:
 

<b>Unlocked</b>	Releases your lock on the files after check-in.
<b>Exclusive</b>	Indicates that you intend to make further changes to the files.
<b>Non-exclusive</b>	Indicates that you are working on the files and may possibly make changes.
<b>Keep current</b>	Retains the current lock status.
4. Continue completing the **Check In** dialog box options as follows.
  - Optionally, check **Force check-in** to check in files regardless of their status.
  - Optionally, check **Delete working files** to remove the selected files from your working folder after they are checked in.
  - (Required if process rules are enforced) Check **Link and pin process item** to link the new files to process items. To use a process item besides the active process item, click **Select** and use the **Select Process Item** dialog box to change the process item.
  - If work on the active process item is now complete, check **Mark selected process item as fixed/finished/complete**.
  - To make changes to the selected process item's properties during the check-in process, check **Show property editor for selected process item**.
  - Optionally, select a **Revision label** from the list, or create a new revision label by typing its name. Existing labels are listed in reverse chronological order, based on the time at which they were created, unless the **Sort View Labels By Name** option has been selected in the **Personal Options** dialog box.
5. Optionally, click **Advanced** to open the **Advanced Options** dialog box.
  - Check **Set EOL check-out format (for text files only)** to control the EOL character stored with the files. The default setting is based on the **EOL** setting in the **File Properties** dialog box.
  - Select an appropriate check-out **File Encoding** from the list.
  - Select the file type of file you have selected: **ASCII**, **Binary**, or **Unicode**.
  - Click **Show Change Requests** to review the change requests linked to the files you are checking in.
  - Click **OK**.
6. Click **OK**.


## Checking Out Files

1. Select the file(s) that you wish to check out from one of the **Eclipse Explorers** or from any view that displays the **Team** context menu.
2. Choose **Team > Check Out** or **Team > Force Check Out** from the context menu.  
 **Note:** Using **Force Check Out** overwrites any files with the same name in your working folder, even if they are more recent.
3. Click **Get incoming changes from StarTeam Server** within a StarTeam view.
4. Choose **Team Synchronizing > Synchronize > Check Out/Force Check Out**. The **Check Out** dialog box does not open, and the files are checked out to your local working folder.

## Checking Out Files with Options

1. Select the file(s) that you wish to check out from one of the Eclipse Explorers or from any view that surfaces the **Team** context menu.
2. Choose **Team > Check Out with Options** from the context menu. The **Check Out** dialog box opens.
3. *Optional:* Check the following options:
  - **Force check out** to overwrite any files with the same name in your working folder, even if they are more recent.
  - **Show check-out statistics**

For a particular check-out operation, you can see how many files are being sent by StarTeam directly and how many are being sent by Cache Agent by displaying the check-out statistics. After the check out operation completes, the **Check Out Statistics** dialog box opens displaying the elapsed time, total number of files, total number of bytes, and whether the check-out operation skipped or failed to check out any of the files.

4. Expand **Reference By**, and select one of the following options for the files you wish to check out:
  - **Current revision:** The most current (tip) revision.
  - **Label :** A specific file revision. The view labels precede the revision labels in the list.
  - **Promotion state:** A specific promotion state.
  - **Configuration as of:** The revision that was the tip revision at the specified date and time. Click the **Date/Year** button to use the calendar, and specify the time by typing in the time or using the spin boxes.
5. Expand **Lock Status**, and select one of the following lock options:
  - **Unlocked:** Does not lock the files you are about to check out, or causes them to be unlocked if you currently have them locked. Be aware that when you check out an unlocked file, it may become read-only in your working folder.  
 **Note:** There is a project property and a Preferences option that controls the read-only or read/write status of working files.
  - **Exclusive:** Indicates that you intend to make further changes to the files.
  - **Non-exclusive:** Indicates that you are working on the files and may possibly make changes.
  - **Keep Current:** Retains the current lock status.
6. *Optional:* expand **EOL Conversion**, and choose one of the following:
  - **None** or another button to change your current EOL conversion setting.
  - For Windows, the EOL marker is **CR-LF** (carriage return/line feed).
  - For UNIX, it is **LF** (line feed).
  - For Macintosh operating systems, it is **CR** (carriage return).

EOL settings on this dialog override the default setting you selected on the **File** page of the **Personal Preferences** dialog box.



7. Expand **Advanced** to optionally select an appropriate **File Encoding** from the list box to support keyword expansion for non-English code pages.
8. In the **Changes** area you can optionally: Show flat, tree, or compressed folder views by selecting the various toolbar buttons in the **Changes** area toolbar.
9. Select a file and click **Compare with Revision** in the **Changes** area toolbar to compare the local file with the tip revision of the file in the repository.



**Note:** The dialog box also provides a *Configure Personal File Options* link so you can quickly make modifications to file options before completing the checkout.

10. Click **OK** to complete the check out.



**Note:** You can also use the **Get incoming changes from StarTeam Server** button on the toolbar to check out files without using the **Check Out** dialog box. If process rules are required, the **Check Out** dialog box opens automatically.

## Checking Out Existing Folders

1. Open the StarTeam perspective.
2. Choose **Window > Perspective > Open Perspective > Other**.
3. Select StarTeam Classic or StarTeam Activity from the list, and click **OK**.
4. Open the Server Explorer, and navigate to the appropriate StarTeam project, view, and folder.  
You can multi-select folders to check out in the Server Explorer.



**Note:** If you have not logged on to the server, right click on the server configuration node, and choose **Log On**. The status bar at the bottom of the Workbench reflects the logged on user name.

5. Right-click and choose **Check Out As Project**. The **Check Out As** wizard opens.
6. Choose one of the following options:

Option	Description
<b>Check out as project in the workspace</b>	Enter a project name in the text box provided or use the default folder name, and click <b>Next</b> . Select a location where you want to check out the files to, and click <b>Next</b> .
<b>Check out into existing project</b>	Click <b>Next</b> to choose a valid target project, and click <b>Finish</b> .
<b>Check out as a project configured using New Project Wizard</b>	Click <b>Finish</b> , and follow the steps in the <b>New Project</b> wizard.



**Note:** If you want to use a location within the workspace that is not the default (that is, you select either the second or third option), then you must use the StarTeam Cross-Platform Client to check out the project and then import it from within the StarTeam Eclipse Plugin.

7. If you chose to **Check out as project in the workspace**, specify any check out options as follows:
  - Expand **Reference By**, and select one of the following options for the files you wish to check out:
    - **Current revision** : The most current (tip) revision.
    - **Label**: A specific file revision. The existing view and revision labels are listed in reverse chronological order based on the time at which they were created. The view labels precede the revision labels in the list.
    - **Promotion state**: A specific promotion state.
    - **Configuration as of**: The revision that was the tip revision at the specified date and time. Click the **Date/Year** button to use the calendar, and specify the time by typing in the time or using the spin boxes.
  - Expand **Lock Status**, and select one of the following lock options:
    - **Unlocked**: Releases your lock on the files after check-in.

- **Exclusive:** Indicates that you intend to make further changes to the files.
  - **Non-exclusive:** Indicates that you are working on the files and may possibly make changes.
  - **Keep Current:** Retains the current lock status. Selected by default.
  - *Optional:* Expand **EOL Conversion**, and choose one of the following:
    - **None** or another button to change your current EOL conversion setting.
    - For Windows, the EOL marker is **CR-LF** (carriage return/line feed).
    - For UNIX, it is **LF** (line feed).
    - For Macintosh operating systems, it is **CR** (carriage return).
  - Expand **Advanced** to optionally select an appropriate **File Encoding** from the drop-down list to support keyword expansion for non-English code pages.
8. Click **Next**. The **Mapping preferences** page opens.
  9. Choose your desired settings, or accept the default settings, and click **Finish**.

## Achieving Consistent Check-ins and Check-outs

Developers can use various StarTeam features to allow or avoid conflicts with other developers on the same files (in the same view). In a low-contention environment, developers can check-out files without locks, modify them, “refresh” to identify and resolve merge conflicts, and then check in modified files. All check-ins in StarTeam are atomic. If more than one file is checked in as the result of a single transaction (for example, in a change package from a View Compare/Merge session) the files and their associated process items are updated in a single action. If for some reason, the check-in fails, none of the files are checked in, and the status of the associated process items is not updated. In general, you can achieve consistent check-ins and check-outs by doing the following:

- Exclusively lock files before checking them in or out and unlock files after a successfully checking them in or out; or
- Temporarily change your view configuration to a known “stable” point

In higher contention environments, developers may want more assurance of getting consistent sets of files during check-out, that is, avoiding files that are a partial set of someone else’s check-in. The easiest way to achieve this need is “by convention”. Each developer exclusively locks all files before checking them in, and unlocks them when they are “complete”, either at check-in or soon thereafter. Correspondingly, each developer exclusively locks all files before checking them out, and unlocks them when complete. If a developer cannot get all the locks at once, they are potentially about to interfere with another developer, so they unlock files they currently have locked, wait a bit (probably talk with the developer they are conflicting with), and then try again. One implication of this “by convention” approach is that a developer could be blocked while waiting for another developer to finish-up.

A more formal way to enforce consistent check-outs is to use “view configuration”. To ensure consistent check-outs without locks, a developer can temporarily change their view configuration to a known “stable” point. In some organizations, a nightly build process creates a view label when the server is not used or lightly used. To temporarily change the view configuration from the StarTeam client, select **View > Select Configuration > Labeled Configuration** and choose the latest build label. (Alternatively, choose a timestamp or promotion state.) The view will switch to show the item states at the selected time, and “consistent” check-outs can be performed from there.



**Note:** The view configuration change is only at the client – the underlying “real” view is not modified. Also, note that “rolled-back” views are read-only: they must be reset to the “current” configuration before new/modified files can be checked-in. An implication to be aware of with this approach is that the time to switch the configuration can take a few seconds to a few minutes on very large views.

## Optimizing File Check-outs Over a Slow Connection

Not surprisingly, one of the most bandwidth-demanding operations is file check-out. Even with compression strategy, checking out a large number of files can require a lot of data transfer.

Most StarTeam users maintain a working set of files on their local disks, allowing them to check out only the new file revisions they need to make the files current. Suppose, for example, that you are working on a

project for which StarTeam tells you that you have 75 "out of date" files. This means that you have an older revision of each file on your local drive, but you need a newer revision to make the files current. Normally, when you select those files and perform a check-out, StarTeam sends you each new file as a "full" revision. For high-speed to broadband network connections, normal file check-out will give you satisfactory performance.

But what if network bandwidth is very tight or congestion is a constant concern? StarTeam provides an option to check out newer file revisions with a different strategy called "delta check-out". Instead of sending full file revisions, delta check-out causes "delta" records to be sent by the StarTeam server that allow the existing revision of each file to be "patched" to the revision you are checking out. Because delta records are usually much smaller than the whole file revision, they require even smaller file check-out messages.

Delta check-out is enabled individually at the client level by choosing **Personal Preferences > File > Optimize for Slow Connections**.

Below are some points to consider in deciding if delta check-outs will help you:

- Delta check-out is possible only with text-based files. Binary files are always checked out by sending full file revisions.
- Delta check-out requires more CPU time and disk I/O than normal (full version) check-out because the existing work file must be "patched" up to the selected version. Hence, you should use this option only when network bandwidth is highly constrained (think modem kind of bandwidth).
- Delta checkouts are not used, nor are they necessary, when you are checking out from a Cache Agent.

## Monitoring Check-out Statistics using the Cache Agent

The visible advantage to using Cache Agent is the improved speed of file check-out operations. The more files you check out, the more advantage you will gain from Cache Agent. Over time, more and more of the files will come from Cache Agent, reducing the strain on StarTeam Server. As a result, the check-out speed should continue to improve until all files are available from Cache Agent.

For a particular check-out operation, you can see how many files are being sent by StarTeam Server directly and how many are being sent by Cache Agent, by displaying the check-out statistics.

1. Select the files to be checked out.
2. Right-click the selected files and choose **Check Out with Options**. The **Check Out** dialog box opens.
3. Check **Show check-out statistics**.
4. Select any other option settings that are appropriate to your check-out operation, and click **OK**.

After the check-out operation completes, the **Check-Out Statistics** dialog box opens displaying the elapsed time, total number of files, total number of bytes, and whether any of the files were failed or skipped during the check-out.

5. Click **OK**.

## EOL Conversion Handling Overview

StarTeam provides support for fixed EOL conversion files. For example, files can be checked out in `LF` format on every platform, regardless of specific options. Also, **Update Status** works for all text files once EOL Format is defined, regardless of what EOL format was used when they were checked-out. For compatibility with older releases of the Cross-Platform Client, if check-out "EOL conversion" is not requested, and EOL Format is `Undefined`, files are still checked out with the EOL conversion with which they were added to the StarTeam Server.



**Note:** The default for automatic EOL conversion for check-out operations is "checked" if the user does not have that option defined already.

### EOL Format Property

The EOL Format Property displays as **EOL Character** in the Cross-Platform Client **Items** pane.

The EOL Format property is only meaningful for text files during the check-out operation.

By default, the SDK will compute the EOL Format when a new text file is added or a new revision is checked in for a text file whose EOL Format is `Undefined`, the file's EOL convention matches the platform default, or EOL Format is set to `Client Defined`. Otherwise, EOL Format is set to the convention found: `Fixed LF`, `Fixed CR`, or `Fixed CRLF`:

- The user can change **EOL Format** to any value (other than `Undefined`) at any time.
- Regardless of their **EOL Format** setting, text files always use a canonical (`CRLF`) format in the vault.

The **EOL Format** property can be manually set in the Cross-Platform Client in the **Add/Check-in** dialog boxes, and the **File Properties** dialog box. The Cross-Platform Client **EOL conversion for add/check-in** options are not available.

## EOL Property Values

The EOL property value is displayed as **EOL Character** in the Cross-Platform Client **Items** pane.

The EOL Format property can be set in the Cross-Platform Client in the **Add/Check-in** and **File Properties** dialog boxes. If selected in the **Add/Check-in** dialog box, StarTeam uses the settings specified in the **File Properties** dialog box.

The EOL Property values are:

<b>Undefined</b>	(null in the SDK).
<b>Client Defined</b>	Causes workstation default or per-checkout EOL conversion option to be used.
<b>Fixed CR, Fixed LF, Fixed CRLF</b>	Causes this EOL format to be used always; the workstation/check-out conversion option is ignored.

## Controlling EOL Characters

You can specify which EOL and path case sensitivity settings are active for file check-out operations that you perform.

1. Choose **Tools > Personal Options** to open the **Personal Options** dialog box.
2. Select the **File** tab.
3. Select or clear **Automatic EOL Conversion On Check-out**.
4. Click the appropriate radio button to specify whether you are working on Windows, Unix, or Mac.

## EOL Format Property

The EOL format property displays as **EOL Character** in file details.

The EOL format property is only meaningful for text files during the check-out operation.

By default, the SDK will compute the EOL format under the following conditions:

- When a new text file is added or a new revision is checked in for a text file whose EOL format is `Undefined`, the file's EOL convention matches the platform default, EOL format is set to `Client Defined`. Otherwise, EOL format is set to the convention found: `Fixed LF`, `Fixed CR`, or `Fixed CRLF`.
- The user can change EOL format to any value (other than `Undefined`) at any time.
- Regardless of their EOL format setting, text files added or checked in always use a canonical (`CRLF`) format in the vault.

The EOL format property can be manually set in the **Check In** dialog box and the **File Properties** dialog box.

## EOL Property Values

The EOL property value is displayed as **EOL Character**.

The EOL Format property can be set in the **Check In** and **File Preferences** dialog boxes. If selected in the **Check In** dialog box, StarTeam Eclipse Plugin uses the settings specified in the **File Preferences** dialog box.

The EOL Property values are:

- **Undefined** (null in the SDK): Used for files added before StarTeam Eclipse Plugin 2009 Release 2.
- **Client Defined**: Causes workstation default or per-checkout EOL conversion option to be used.
- **Fixed CR**, **Fixed LF**, and **Fixed CRLF**: Causes this EOL format to be used always; the workstation/checkout conversion option is ignored.

## Process Items and Process Rules

This section discusses the use of process items and process rules.

### Process Items

Modern development practices require increased control over the entire development process. StarTeam enables developers to follow a defined development process, one that ensures that all file content changes be linked to either a change request, requirement, task, or custom component. Items used in this way are known as *process items*.

Specifically, a process item is a change request, requirement, task, or custom component that is specified by the user as the reason for making a given set of changes. Process items are supported by the **Add Files** and **Check In** dialog boxes. As a result, source code and content are modified only to meet clearly defined and approved objectives, as expressed in the process item.

#### Out-of-view Process Items

Historically, StarTeam has supported the selection of a process item from only within the current view. This functionality is useful in many processes, but it does not support a process where change requests, tasks, requirements, or custom components live in a different view than the source code files.

To support out-of-view process, StarTeam now enables you to choose a valid process item for file add or check-in operation from any view on the same server as the files being committed. You can choose an item selected in the **Items** pane as the active process item for the current view, an open view on the same server, or a different view on the same server.

Also, the **Active Process Item** toolbar button contains a list enabling you to select the active process item from any opened view.

#### Process Item Selection

StarTeam currently allows a process item to be selected as the active process item, which results in that process item being used by default in the **File Add** and **File Checkin** dialog boxes. The **File Add** and **File Checkin** dialog boxes also allow you to change the active process item prior to adding or checking in the files.

## Process Items and Workspace Change Packages

Process items act like lightweight change containers. Using process items enables you to link and track changes made to your files, even when you and other members of your development team are not required to use process rules. They provide traceability, allowing you to trace file changes to their purpose or

context. They also provide a way to identify file revisions for a specific change request, task, or requirement so that, for example, you can attach those revisions to a view or revision label.



**Note:** The creation of process tasks and the *Enhanced Process Model* has been replaced by workspace (check-in) change packages. Files linked to a process item can now be viewed in the details of the **Change** tab or view.

Workspace (check-in) change packages are created when the following actions are performed on files and folders:

- Add and check-in
- Move
- Rename
- Delete

When files are committed using a process item, the process item is linked via a trace to a workspace (or check-in) change package representing the atomic commit of files.

Specifically:

- The process item can be a change request, a task, a requirement, or other custom component defined for **Process Rules** in the **Project Properties**.
- The process item can live in any view. It does not necessarily have to reside in the same view where the changes are being performed.

The StarTeam administrator can enforce the use of process items for a project by establishing process rules. You define process rules in the Project Properties dialog box. Process rules specify that process items must be used when checking in files, and they establish which type of items can be used as process items.

When process rules are enforced, you must link and pin all files you add or check in to a process item. If process rules are not enforced, you can still take advantage of the linking and tracking made possible with process items. As you add files or check them in, you indicate that the new file revisions are to be linked and pinned to a specific process item. You do this by selecting a change request, requirement, task, or custom component as the process item for the operation. At the same time you can mark the change request as fixed, the requirement as complete, or the task as finished.



**Note:** If there is an active process item available, the **Check In** dialog box automatically fills in the **Process Item** field.

Using process items enables you to clearly distinguish the following:

- Which file revisions are related to or fix a specific change request.
- Which file revisions are related to or complete a specific requirement.
- Which file revisions are related to or finish a specific task.

Each view can have a different active process item. As you change from view to view, the process item information displayed on the status bar changes.

## Process Rules

Each project has the option of enforcing the use of process items by specifying certain *process rules*. When enforced, the process rules require you to specify a specific process item (change request, requirement, or task) for file add or check-in operations within the project.

### Process Rules Review

If process rules are not enforced, any change request, requirement, or task can be used as a process item, regardless of its status. However, if process rules are enforced, you may be able to select only one type of item as a process item. In addition, acceptable process items may be limited to those with specific statuses.

You can determine whether process rules are in effect for a specific project, and what those rules are, by reviewing project properties. If you do not have the access rights necessary to do this, ask your administrator what process items apply to the project and what restrictions have been placed on them.

To set process rules, you must have the access rights required to change project properties. As a rule, only team leaders and administrators have these rights. To use process items, project users must have the necessary access rights, which are the rights to:

- See and modify the types of items used as process items in the project view.
- Create and modify traces.
- View and create change packages.

### Advantages of Process Rules

Establishing a system of process rules allows you to:

- Require that process items be used every time files are added or checked into the project.
- Stipulate that only certain types of items with specific statuses can be used as process items in the project.

If process rules are not enforced, linking and pinning to a process item during file add and check-in is optional, and you can select any change request, requirement, or task as a process item, regardless of its status.



**Note:** As a convenience, you can select a change request, requirement, task, or other custom component that has been established as a valid process item type, as the *Active Process Item* prior to adding or checking in files so the check-in automatically use that process item during check-in.

## Active Process Items

The Client enables you to pre-select a process item as the active process item to use the next check-in operation. This type of process item is referred to as an *active process item*.

You can select a change request, requirement, task, or custom component as an active process item before adding or checking in files. Pre-selecting an active process item is a convenient way to save time when you know that you will be adding files or checking them in later using the designated process item. When you have a process item selected on the upper pane, making it the active process item is a simple operation. Before or during each check-in operation, make sure you select the correct *active process item*.

The active process item is the default selection when you add files or check them in. However, you can change your mind and select another appropriate item. An active process item is used until its status becomes ineligible or another process item is chosen. Additionally, if an active process item is available in the StarTeam Cross-Platform Client at the time of a move or delete of files, then a workspace change package is created for the transaction and a trace is created from the active process item to the workspace change package.

The **Status** bar displays the name of the active process item. If the active process item is from a different view and project than the current view, you can hover over the active process item shown in the status bar and see a tool tip showing the project and view as well as the process item description. The **Status** bar pre-pends the name of the active process item with the name of the project and view. To see details about a process item, you can double-click the item in the status bar to open the Properties dialog box with more information.



**Note:** You can only specify one item for each view as an active process item. Selecting a second active process item clears the first.



**Tip:** After you finish with a process item, you should choose **Clear Active Process Item** on the menu so that it cannot be accidentally reused. That removes the information from the status bar and keeps the process item from reappearing in the **File Add** or **File Checkin** dialog boxes.

## Promoting File Changes Into Baselines

Process rules are useful when creating baseline builds or configurations. A build is a labeled configuration that identifies the file revisions and process items that define the code and content baseline. Process rules require that each new file revision be linked to a process item, which allows the development team to promote these changes into baselines.

If process rules are not enforced, developers using the application can create baselines either by:

- Labeling an entire project view at a specific point in time.
  - Associating file revisions with a revision label on check-in.
1. Start with the previous baseline (for example, check it out based on its label.)
  2. Select process items for inclusion in the new baseline.
  3. Label the new baseline You can use process items for tracking purposes when adding or checking in files.

## Displaying Only Enhanced Process Links

If your project previously had enhanced process links enabled, you can choose to display only enhanced process links in the **Link** pane.

1. Click the **Link** tab in the StarTeam perspective.
2. Right-click the arrow in the **Link** tab toolbar and choose **Show enhanced links only**.

## Filtering Process Tasks From Other Tasks

If you have previously enabled enhanced process links in your project so that StarTeam created process tasks, you can filter your tasks to separate the process tasks from the regular tasks. Use the **Usage** field value to determine the difference between process tasks and standard tasks. If the **Usage** value is anything other than **Other**, then it is a process task.

1. Choose **Filters > Filters** in the **Task** view of a StarTeam perspective. This displays the **Filters** dialog box.
2. Click **New** and give the new filter a name.  
*Alternative:* Copy an existing filter by selecting it, clicking **Save As**, and giving it a new name. Then select the copied filter and continue with the next steps.
3. Click **Fields**.
4. Move the **Usage** field from the **Available Fields** list to the **Show Fields in this Order** list.
5. Click **OK**.
6. Click **Query**. This opens the Edit Query dialog box. The **Queries** dialog box opens.
7. Click **New**. The **Edit Query** dialog box opens
8. Type a **Name** for the new query.
9. Choose the following in the **Condition Node** section:
  - **Field** = Usage
  - **Operator** = Not Equal
  - **Value** = Other
10. Click **Add** to add the condition to the query.
11. Click **Save** to save the query. Returns to the Queries dialog box. Your new query is now highlighted in the list of queries.
12. Click **Select** in the **Queries** dialog box use this query in your new filter. You are returned to the **Filters** dialog box, and your new filter should be highlighted.



13. Click **Save As** to save the filter.



**Note:** Conversely, you can create a filter that displays only the standard tasks. In the query, use the condition **Usage Equals Other**.



**Tip:** If you use tasks on a regular basis and not just for process tasks, add **Usage Equals Other** to existing queries so you never see process tasks when working on tasks that have been manually created, or imported from Microsoft Project using Microsoft Project integration plugin.

## Setting Active Process Items

Setting an active process item is a convenient way of saving time when you know that you will be adding files or checking them in later.

You can make a selected change request, task, or requirement the active process item for the current view, an open view on the same server, or a different view on the same server.



**Note:** You can only specify one active process item for each view. Setting a second active process item for the same view at one time clears the first one.

1. Open your project in a StarTeam perspective.
2. Click the **Change Request**, **Requirement**, or **Task** tab and select the item you want to use as the active process item.
3. Do one of the following:
  - Right-click and choose **Set Active Process Item > Current View** to choose the current view.
  - Right-click and choose **Set Active Process Item > [view name]** to choose from the listed opened views on the server.
  - Right-click and choose **Set Active Process Item > Select View** to open a dialog box and choose any other view on the server.



**Note:** You can also set the currently-selected item as a process item by using the **Use As Active Process Item** button on the toolbar.

The active process item you selected is used by default when you add files or check them in. However, you can override this default and select another appropriate item when adding or checking in files.



**Tip:** After you finish with a process item, you should right-click it and choose **Clear Active Process Item** so that it cannot be accidentally reused. That removes the information from the status bar and keeps the process item from reappearing in the **File Add** or **File Checkin** dialog boxes.

## Establishing Process Rules for Projects

Establishing a system of process rules allows you to:

- Require that process items are used every time files are added or checked into the project.
- Stipulate that only certain types of items with specific statuses can be used as process items in the project.
- Enable the use of enhanced process links for the project.



**Note:** To set process rules, you must have the access rights required to change project properties. Usually, only team leaders and administrators have these rights. You must also verify that project users have the rights to see and modify items in the project view, to create and modify links on files and process items, and to create tasks and link to tasks if using the enhanced model.

1. Choose **StarTeam > Project > Properties**. In the Server, Navigator, or Package Explorers, right click on the project, and choose **Properties** from the context menu. The **Properties** dialog box opens.
2. Expand **StarTeam Provider**, and select **Project**.
3. Select the **Require Selection Of Process Items When Files Are Added Or Checked In** check box.

4. Select the type you want to allow for use as process items.
5. You can define the use of the type as a process item in the **Process Item Details for <Type>** section.

To permit the use of any type as a valid process item:

1. Select the desired **Type**.
2. Specify the **Active States** that are permitted to be used as a process item during commit.
3. Specify the **Closed State** that will be used to mark the process item as completed upon successful check-in.
4. Add the <Type> as a valid process item type.



**Note:** Some StarTeam integrations do not recognize process rules and will ignore them.

## File Compare/Merge

File Compare/Merge is a graphical file and folder comparison and merge tool delivered with StarTeam. It enables you to compare the contents of two files or folders, and manually or automatically merge the contents. The File Compare/Merge panes highlight differences using a configurable color scheme, and dynamic action buttons display in the highlighted areas to simplify the merging process.

There are three versions of File Compare/Merge, and how you start File Compare/Merge determines which features are available. The ability to edit text files in a File Compare/Merge pane depends on which version of File Compare/ Merge you are using, and what type of files you are comparing and merging.



**Tip:** In **File Preferences**, you can specify File Compare/Merge as the alternate compare/merge utility to use instead of the default tool.

The following list describes editing capabilities in each version of File Compare/Merge.

File Compare/Merge Version	Capability
Main File Compare/Merge	Edit a local file, a merged base file, and a copy of a repository revision.
Embedded File Compare/Merge	No editing is possible.
Standalone File Compare Merge	Edit all files being compared and merged.



**Note:** You cannot edit the actual historical revision of a file in the StarTeam repository.

## Main File Compare/Merge

The main File Compare/Merge window is displayed using menu commands in the client, and it opens in a separate window enabling you to do the following:

- Compare the contents of a local file with the tip revision stored in the StarTeam repository. You can also edit the contents of the local file from within the File Compare/Merge window, and save the changes for check-in.
- Compare two revisions of a file listed on the **History** view tab in the StarTeam Eclipse Plugin application window. Editing the content of historical revisions is not allowed.
- Merge the contents of a local file with the tip revision in the StarTeam repository. The merge results are stored locally, and the file status is changed to **Modified** in StarTeam so you can check in the file.
- Edit the temporary local copy of a repository revision and save it as a file with a different name.

The main File Compare/Merge gives you the option of viewing a third pane for displaying the merge results. You can edit the contents in the third pane and save your merged results.

You can start File Compare/Merge using context menus on selected files in the StarTeam Eclipse Plugin, or by checking in an older version of a file which causes a merge situation.

## Embedded File Compare/Merge

The embedded File Compare/Merge gives you a quick way to do a comparison of text in two files or versions of the same file, as well as compare properties of non-file items such as change requests. If you are comparing two text files, it performs a dynamic comparison of two selected files, or the selected repository file with your local working copy. It displays the text contents of both files in an embedded pair of panes at the bottom of the StarTeam window. These embedded panes only compare the contents of two text files, and do not permit editing or merging.

If you are comparing the properties of two non-file items, only the property values of each selected item display in the embedded panes.

When the embedded File Compare/Merge is activated, it immediately compares the files or properties when you do one of the following:


- Select a file or other item in the upper pane, or the **History** view. The comparison is between the local working copy of the item and the selected item revision in StarTeam.
- Select two items in the **History** view. The comparison is between the two historical revisions in StarTeam.

## Standalone File Compare/Merge

The standalone File Compare/Merge is started outside of the StarTeam Eclipse Plugin from the Windows **Start** menu. This File Compare/Merge compares files, folders, and images, and it can merge the contents of two text files or two folders. Unlike the main File Compare/Merge, the standalone version does not compare any local files with files that are in the StarTeam repository. It compares and merges two or three files, folders, or images that are on your local computer or network. You can edit text file contents directly in the File Compare/Merge panes, and you can move lines or blocks of text between the panes. You can also move folders between the panes during a folder comparison or merge.

## Specifying FCM as Alternate Compare/Merge Tool

Post installation, the Eclipse compare/merge tool is selected as the default comparison/merge utility. If you want to use File Compare/Merge as your alternate compare/merge tool instead, you must configure those settings in **Preferences**.

 **Note:** Refer to the Release Notes for information on installing the File Compare/Merge application.

1. Choose **Window > Preferences** from the main menu.
2. In the **Preferences** dialog box, expand **Team > StarTeam > File**.
3. In the **File** panel, select **Use StarTeam compare/merge**.
4. Select the applicable check box for one or both application types you want to use: **Merge utility** and/or **Comparison utility**.
5. Type or browse to the path for the executable file for each selected File Compare/Merge application.
6. In **Options**, type any options you want to use for the selected applications
7. Click **OK**.

## Labels and Promotion States

This section contains information related to managing labels and promotion states.

# Labels

In version control, the term `label` corresponds to the act of attaching a view or revision label (name) to one or more folders/items. StarTeam enables you to create two types of labels:

**View labels** Are automatically and immediately attached to all folders and items in a view at the time you create the view label. View labels have multiple purposes, but you primarily use them to place a *time stamp* on the entire contents of the view and as *build labels*. When you roll back the view to that label, you see everything that existed at that point in time—unless the label has been adjusted. You can create a view label for a specific point in time or as a copy of another existing view label. Unless the view label is frozen, you can adjust it to include or exclude some folders and items by attaching or detaching view labels. You can also move a view label from one revision to another.

**Revision labels** Are not attached automatically to any item in the view. Instead, they are used to designate a set of folders or items within a view. For example, you might want to label a group of files that should be checked in and out together. After you create a revision label, you attach specific items, building it up to reflect a specific set that is typically a small subset of the view. StarTeam can automatically attach new file revisions to a revision label at check-in time if you like.

## About Labels

You can attach a label to any type of StarTeam item, including folders, files, requirements, change requests, tasks, topics, and audit entries. Any item can have more than one label. However, no two revisions of the same item can have the same label at the same time.

Every label is unique within its view. That is, no view label can have the same name as any other view label, no revision label can have the same name as any other revision label, and no view label and revision label can have the same name. Each view has its own set of labels. This also means that every label has a *name* that must be unique from other labels belonging to the same view. Each label also has a *description* that helps users understand the purpose of the label.

You can manually attach or detach both view labels and revision labels to or from a folder or item. In addition, you can use either type of label to identify a file when it is checked out. When you check a file in, you can attach and create a revision label for that file or attach an existing revision label.

You can select any type of item by its label. For example, you can select all files with a particular revision label and roll them back to that label, making the revision with that label the tip revision. Then you can compare your working files to the rolled-back revisions.

You can set access rights for labels at the view level or at the folder or item level. You must grant the rights to create labels, edit their properties, and delete them at the view level. However, you can grant the right to move a label (also called *adjust a label*) at the folder or item level.

A label can be frozen, which means no new artifacts can be attached to it, and attached artifacts cannot be detached nor reattached at a different revision. Conversely, non-frozen labels can have all of these modifications. Since many organizations depend on the immutability of frozen labels, a specific security permission is required to freeze or unfreeze a label.

StarTeam actually attaches specific *item revisions* to a label, each of which infers a specific artifact revision. Since each item specifies a parent folder, the artifact is attached in a specific folder. This means that if you move an item, you need to reattach it to any label for which you want to reflect the artifact in its new folder. If you don't reattach a moved item to an existing label, it will continue to be attached to the label in its old folder (which may be what you want).

Even if frozen, both view and revision labels can be cloned. That is, you can create a new label starting out identical to an existing label and then adjust the revisions attached to it. A common practice is to clone a previous label, attach only new file revisions that were created to fix a bug, and use the new label to identify the file revisions for a new build candidate or release candidate.

## Time Stamps and Build Labels

Using a view label as a *time stamp*, you can roll a view back to see everything in the view as it was at the time the label was attached. For example, to see if a particular file was in the beta version of a product, you can roll back the view to the beta label.

You may also use a view label as a *build label*, which allows the QA team to immediately determine what build to test for a fix to any given change request. To use a view label for this purpose:

- It must be designated as a build label.
- It must be created while the **Addressed in build** property for the change request contains the value `Next Build`.

When StarTeam creates the label, each change request with Next Build as its **Addressed in build** property will be reset to the build label.

To create a view label, you must select the current configuration of the view. Historical configurations are read only, and adding a label is considered a change. However, if a label already exists for a prior configuration, you can adjust its name, files and folders can be added to it or detached from it, and so on. You can also move a view label from one revision to another.

For example, suppose your administrator creates a view label before each build, giving that label to the tip revision of each file in the view, and then checks out all the files with that label for the build. If the tip revision of one file does not change for a few weeks (or longer), it can acquire several view labels, while a file that changes frequently may have several revisions with no view labels and other revisions with only one view label.

When you detach a view label from a folder, StarTeam automatically detaches the label from everything in the subtree for which the folder is the root. If you roll back a view to a specific view label and a folder does not have that label, you cannot see the children of that folder and their contents anyway.

You can only create a view label at the view level and only while the configuration is current. However, you can create a view label for the current configuration or for a time in the past. In either of these two cases, StarTeam attaches the new label to the tip revision of each folder, file, change request, task, or topic that belonged to the view at the specified time.

You can also create a view label as a copy of an existing view label or as a copy of the view label currently attached to a promotional state. In these two cases, StarTeam attaches the new label to exactly the same items and revisions as the existing view label.

You can check file revisions out using this label or roll back the view to this label and see all the items associated with that label. For example, if you create the view label *Build 100* as you make Build 100 of your product from a view, all the files in the view will have the label *Build 100*.

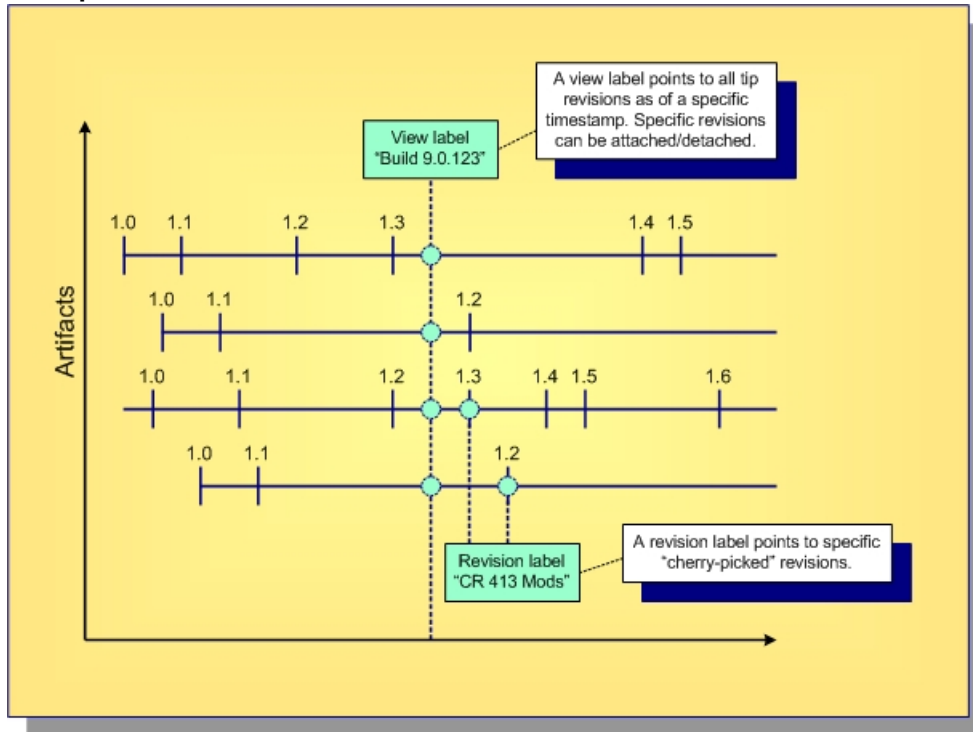
If some items should not be included, you can detach the label from those items individually. For example, if some files should not have that label, select the files then select **Labels > Detach** from the **File** menu or context menu to detach that label. If the files that should not be included all belong to the same folder and are the only files in that folder, use the **Labels** command on the **Folder** menu. For example, if the help files were not checked in until after the view label was attached, you can move that label from the previous revisions of the help files to the newly checked-in help files.


## Label Access Rights

You can set access rights that apply to labels at the view level and at the folder/item levels. You set the access rights that allow a user or group to create labels, edit their properties, and delete them at the view level. For example, if you can create a label, you can set its initial properties. However, if you do not have the right to edit label properties, you cannot later freeze or unfreeze that label.

You can attach labels to individual folders or items, detach from them, or move from one of their revisions to another. The access right to move a label is named **Adjust a label**. You can grant or deny these rights at the folder or item level.

## Example View and Revision Labels



 **Note:** Not all items in a view have to be attached to a label. Conversely, an item can be attached to any number of view and revision labels as you like, but only one revision of an item can be attached to any specific label.

## View Labels

View labels should be used to mark configurations of the entire view that match specific milestones such as a new build number, a point from which another view was created, a release candidate, and so forth. Consequently, you don't create or use view labels for specific check-ins. Instead, you create them when one or more changes have been committed and it's time to launch a new build, spawn a release view, or promote changes to another view. The following two scenarios illustrate good uses of view labels.

The following two scenarios illustrate good uses of view labels.

### Scenario 1: Daily Builds

Whether your team prefers daily builds, nightly builds, hourly builds, or builds on demand, this approach creates new view labels where "Use as build label" is selected:

- To get the process going, first create a view label as of a specific timestamp and launch the build process. (Some users prefer to have the build process itself create the view label.)
- The build process then checks out files attached to that label and launches compiles, links, and other build tasks. If these tasks are successful, the same build process could also launch unit tests and other verification tasks.
- If all build tasks are successful, the view label could be marked as "frozen", which identifies it as a "good build".
- Conversely, if a build task fails, the build process could generate a report or notify someone via email. If you're early in the development activity, you might choose to just move forward, allowing the team to continue making changes in the view. The next build process will simply "try it again". When you're later in the development activity, you'll want to have the appropriate developers fix their errors, re-attach the new revisions they create to the same view label, and perform the build again. Only when you achieve a "good build" is the view label marked frozen.

The advantage of this approach is that it tends to ensure that the tip revisions in a view are generally buildable. This supports a growing software development practice known as *continuous integration*. The disadvantage of this approach is it may be difficult with large teams and environments with lots of changes. It can result in a lot of broken builds, finger pointing, and “nasty gram” emails.

## Scenario 2: “Change” Builds

Instead of relying on tip revisions being generally stable and buildable, another approach is to create view labels that are attached to revisions that are carefully selected. The steps that take this path are outlined below:

- Assume you start with an existing “good build” view label. As with the previous scenario, this label would be flagged as a build label and probably frozen.
- Although many changes are occurring in the build, you want to select only specific changes as candidates for inclusion in the next “good build” label. To do this, ensure that the corresponding file revisions are attached to a revision label and that this label is **only** attached to the file revisions you’re interested in.
- Start the next “good build” label by “cloning” the current label. In the StarTeam Cross-Platform Client, select **View > Labels > View tab > New**. In the corresponding dialog, choose **Labeled configuration** and select the current `good build` label. Your new label will now be identical to the old label.
- Select the file revisions associated with the desired changes. In the **File** tab, select **Select > By Label** and choose the appropriate revision label.
- Now attach these revisions to the cloned view label. In the **File** tab, select **Labels > Attach**. In the corresponding dialog, select the cloned label in the upper label list. In the “Attach to items at” group box, select “Labeled configuration” and choose the same revision label you used in the previous step. This ensures that the correct revision of each file is attached to the cloned label, otherwise the tip revision will be attached, which may be the wrong revision.
- Repeat the previous two steps if there are multiple revision labels representing file revisions that should be included in the new label.
- Now launch the appropriate build and test process for your new “good build” label. Mark the label frozen or reattach fix revisions and retry as in the previous scenario depending on whether the build/test process succeeds.

This approach allows you to tag view configurations as candidates for builds, promotes, etc. without relying on the tip revisions being stable. The disadvantage of this approach is that if your latest “good build” label is way behind the view’s tip configuration, the quality of more recent changes may not be known for a while (which goes against the premise of continuous integration.)

Because this approach employs label “cloning”, there is a caveat we should mention with respect to deleted items. Suppose an item is attached to a view label and then deleted from the view. As you’d expect, when you “time travel” by adjusting your view window back to the view label, the item “reappears” because it existed when it was attached to the label. Less obvious, however, is that when you clone the label, the item will also be attached to the new label because the new label is initially identically to the old label. If you don’t want items deleted in the tip configuration to be attached to a cloned label, just detach them from the cloned label.

## Creating View Labels

View labels, usually used as build labels by default, can be extremely useful when you want to label every folder and item in a particular view.

1. With a StarTeam perspective or view open, choose **StarTeam > View > Labels** or **StarTeam > View > Properties** from the main menu. The **Properties** dialog box opens.
2. In the **Properties** dialog box, do one of the following:
  - If opened from the Navigator or Package Explorers, expand **StarTeam Provider** and select **Labels (for view)**.

- If opened from the StarTeam main menu or Server Explorer, select **Labels**.

This **View** tab lists existing view labels in reverse chronological order, based on the time at which they were created.

3. Click **New**. The **View Label** dialog box opens.
4. Type a name and description for the label. The maximum name length is 64 characters and the description length is 254 characters.
5. Select one of the following:
 

<b>Current Configuration</b>	Attaches the label to the tip revision.
<b>Labeled Configuration</b>	Attaches the label to the revision with a specified label. The labels are in reverse chronological order based on the time at which they were created.
<b>Promotion State Configuration</b>	Attaches the label to the revision currently in a specified promotion state. (Actually, the label is attached to the revision that has the promotion state's current view label.)
<b>Configuration As Of</b>	Attach the label to the revision that was the tip revision at a specified date and time.
6. Optionally, check **Use As Build Label** to update each change request that has **Next Build** as the setting for its **Addressed In Build** property. If this option is not selected, the view label will still be attached to change requests, but the setting of the **Addressed In Build** property will not change.
7. Optionally, to freeze the label so that the revisions attached to it cannot be changed, check **Freeze**.
8. Click **OK**.



**Note:** It is always important to synchronize the dates and times of the computers that run the StarTeam clients and the StarTeam Server. However, if they are not synchronized and you select the current time as a label's configuration, the label may not be immediately visible.

## Creating Revision Labels

Like view labels, new revision labels can be created from the **View** menu. In fact, if you are creating a new revision label based on an existing revision label in another view, you must use the **View** menu for this purpose. However, revision labels can also be created from the **Folder Tree** menu, a component menu, or the context menu.

1. Select the folder in the Server Explorer or the Eclipse Explorers that contains the item to modify.
2. Select an item in the Server Explorer or in one of the Eclipse Explorers.
3. Choose **StarTeam > View > Labels** from the main menu.

The **Properties** dialog box opens.

4. Click the **Revision** tab. The labels are listed in reverse chronological order based on the time at which they were created.
5. Type a name and description for the label. The maximum name length is 64 characters and the description length is 254 characters.
6. Optionally, check **Frozen** to freeze the label so that revisions attached to it cannot be changed.
7. Click **OK**.

## Configuring or Viewing Label Properties

View label properties include a name, description, frozen/unfrozen status, configuration, and build label status. Revision label properties include a name, description, and frozen/unfrozen status.

1. Select the folder in the Server Explorer or the Eclipse Explorers that contains the task that you want to modify.



2. Choose **StarTeam > View > Labels** The **Properties** dialog box opens.
3. Select the **Labels** node.

If you open the **Properties** dialog box from one of the Eclipse Explorers, you must expand the **StarTeam Provider** node first, then select the **Labels** (per View) node.

4. Select a label on the **View** or **Revision** tabs. .
5. Click **Properties**. The **Edit Label** dialog box opens enabling you to modify the label name or description, and freeze or unfreeze the label.

## Attaching Labels to Folders

Labeling folders is slightly different from labeling items. When you attach a revision label to a folder, you can also attach it to the items that the folder contains and to everything in the subtree for which the folder is the root (its child folders and their contents).

If you detach a revision label from a folder, you can also detach the label from the items associated with the folder and, optionally, from the child folders and their items. If you detach a view label, the label is automatically detached from the items that the folder contains, from the child folders, and from their contents.



**Note:** To determine whether a label is a revision label or a view label, double-click the label (or select the label, and then click **Properties**). A revision label has a name and a description. A view label has a name, description, and a configuration time.

## Creating a New Revision Label and Attaching it to a Folder and its Contents

Determine whether a label is a revision label or a view label by opening the Label view, and double-click the label (or select the label, and then choose **Properties** from the context menu). A revision label has a name and a description. A view label has a name, description, and a configuration time.

1. Select a folder in the Server Explorer or from one of the Eclipse Explorers.
2. Choose **StarTeam > Folder > Labels**. The **Properties** dialog box opens.
3. Right-click the folder and choose **Labels** to open the **Labels** dialog box.
4. Select the revision that will receive the new label.
5. Right-click the selected item(s) and choose **Labels > New** . The **Attach a New Revision** dialog box opens.
6. Type a name and description for the label. The maximum name length is 64 characters and the description length is 254 characters.
7. Optionally, check **Frozen** (that is, cannot be changed) to ensure that only the selected revision can have this label.
8. Select one of the following:

<b>Folder Only</b>	Attaches a label to the selected folder.
<b>Folder and Items Contained in Folder</b>	Attaches a label to the folder and its items.
<b>Everything in Subtree Rooted at Folder</b>	Attaches a label to the folder, its items, and its child folders and their items.



**Note:** Because attaching a label to a folder also allows it to be attached to the folder contents, children, and so on, the label is always attached to the current configuration of each folder and item. You cannot label a prior revision of a folder.

## Attaching an Existing View or Revision Label to a Folder and its Contents

1. Select the folder in the Server Explorer or the Eclipse Explorers that contains the item to modify.
2. Choose **StarTeam > Folders > Labels**. The **Properties** dialog box opens.
3. Click **Attach**. The **Attach a Label** dialog box lists all the existing labels and identifies them as view or revision labels. By default, both the **View Labels** and **Revision Labels** check boxes are selected.
4. To display only view labels or revision labels, uncheck the appropriate check box.
5. Select a label.
6. Select one of the following:

<b>Folder Only</b>	Attaches a label to the selected folder.
<b>Folder and Items Contained in Folder</b>	Attaches a label to the folder and its items.
<b>Everything in Subtree Rooted at Folder</b>	Attaches a label to the folder, its items, and its child folders and their items.

7. Click **OK**.



**Note:** Attaching a label to a folder always attaches it to the current configuration of each folder and item. It is not possible to label a past revision of a folder, although you can do so for items.

## Attaching Labels to Items

If you are dealing with an item or set of items that you want to group together, you can create a new revision label, attach an existing label to the item or an item revision, review all labels, or move a revision label.



**Note:** A Label can be attached to only one revision of an item.

## Attaching an Existing View or Revision Label to Selected Items

1. Select one or more items in the Server Explorer, any StarTeam Cross-Platform Client view, or in the Eclipse Explorers.
2. If selecting items from the Server Explorer or a StarTeam Cross-Platform Client view, right-click and choose **Labels > Attach**. If selecting items from an Eclipse Explorer, right-click and choose **Team > Labels > Attach**. The **Attach a Label** dialog box opens. By default, both the View Labels and Revision Labels options are checked.
3. Select a label from the list.
4. Indicate what item revision is to receive this label by selecting a configuration option. The choices are:

<b>Current Configuration</b>	Attaches the label to the tip revision.
<b>Labeled Configuration</b>	Attaches the label to the revision with a specified label. The labels are in reverse chronological order based on the time at which they were created.
<b>Promotion State Configuration</b>	Attaches the label to the revision currently in a specified promotion state. (Actually, the label is attached to the revision that has the promotion state's current view label.)
<b>Configuration As Of</b>	Attach the label to the revision that was the tip revision at a specified date and time.

5. Click **OK**.

## Demoting View Labels

Sometimes a labeled set of files is promoted prematurely and must be demoted. For example, if a specific build is promoted to the `Beta` state, but contains serious flaws, it should probably be returned to the prior promotion state. You can only demote view labels by editing the promotion state.

1. With a StarTeam perspective or view open, choose **StarTeam > View > Promotion** or **StarTeam > View > Properties**. The **Properties** dialog box opens.
2. Click **Promotion**.
3. Click **Edit**. The **Promotion State** dialog box opens.
4. Select a different view label from the **View Label** list.
5. Click **OK**.

## Promoting View Labels

You can promote a view label from one promotion state to the next if you have the appropriate access rights.

1. Click **StarTeam > View > Promotion** or **StarTeam > View > Properties**. The **Properties** dialog box opens.
2. Click **Promotion**. The states are displayed from the final state down to the initial state.
3. Select the promotion state currently associated with the view label that you want to promote.
4. Click **Promote**. The **Promote View Label** dialog box indicates that the view label is now associated with the next state (the state immediately above the selected state in the **Promotion** dialog box).
5. Verify that the information is what you were expecting to see.
6. Click **OK**. The selected view label now applies to two promotion states: the one to which it was promoted and the one you originally selected. Usually, your next action is to associate a new view label with the original state.

## Copying Revision Labels

Occasionally, you may want to copy a revision label. For example, if you move or share an item from one view (source view) to another (target view), labels from the source view do not become part of target view. However, by copying the revision labels after the move or share, you can selectively maintain revision labels on the moved or shared items.

Copying a revision label immediately attaches it to the same revisions of the same items as the original revision label. If the two revision labels are in the same view, each label will be attached to the same number of items. However, if the two revision labels are in different views, the new label becomes attached to the same revisions of the same items only if the items and their revisions exist in the new label's view at the time of the copy operation.

Although you can copy revision labels in a variety of ways, the following procedure allows you to copy a revision label whether it is in the current view or in another accessible view. It assumes that you are dealing with files, but can be adapted for other types of items.

1. Select the folder in the Server Explorer or the Eclipse Explorers that contains the item to modify.
2. Choose **StarTeam > View > Labels** from the main menu.

The **Properties** dialog box opens.

3. Click the **Revision** tab.
4. Click **New**. The **Revision Label** dialog box opens.
5. Type a name and description for the label. The maximum name length is 64 characters and the description length is 254 characters.

6. Check **Copy From Another Revision Label**.
7. Click **Select** to open the **Copy a Revision** dialog box.
8. Select a project from **Project** from the list, a view in the **View** tree, and a revision label in the **Labels** list.
9. Click **OK**.
10. Click **OK**.
11. Click **Close**.

The new revision label is attached to the same revisions as the existing label.

12. Do one of the following:

- Check in the changed file or files using the new revision label.
- Check in the changed file and attach the new revision label manually to the changed file revisions. To do this, select the checked in file and click the **Label** tab in the lower pane. Drag the new revision label to the correct (probably tip) revision in the **Label** pane. Repeat for any other changed files.



**Note:** If you have added a new file, use **File > Labels > Attach** to attach the label.

## Deleting Labels

In StarTeam, you can completely remove a view or revision label from a view, although you can create a new label with the same name later, if desired. When you delete a label, it is no longer visible in any list of labels, nor is it attached to any folders or items.



**Note:** If a label is frozen, you must unfreeze it before you can delete it.

1. Select the view in Server Explorer for which you want to delete a label.
2. Choose **StarTeam > View > Labels** from the main menu.

The **Properties** dialog box opens.

3. Select the tab for the type of label you are deleting, the **View** tab for a view label, or the **Revision** tab for a revision label.
4. Select the label and click **Delete**. This removes the label from the view.

## Detaching a Label from a Specific Revision


If you decide not to include certain items in a view or revision label, you can detach the label from those items individually or as a group. Generally, the items from which labels are detached are files or folders.

To detach a label from a specific item revision

1. Select a folder in the Server Explorer or the Eclipse Explorers.
2. Select one or more items in the Server Explorer or in one of the Eclipse Explorers.
3. Open the **Label** view.
4. Click **Link with Selection**.
5. Expand a revision in the **Label** pane to see all labels attached to it as children of the revision.
6. Right-click the label you wish to remove and choose **Detach**.
7. Click **OK**.


## Detaching Labels from Folders

If you decide not to include certain folders in a view or revision label, you can detach the label from those folders.

 **Note:** You can attach and detach any labels from items in current view configurations, but you cannot see deleted items in those configurations. You can detach view labels from deleted items only if you roll back the view to a configuration based on the label you want to detach.

To detach a view or revision label from a folder and its contents:


1. Select a folder in the Server Explorer or the Eclipse Explorers.
2. Choose **StarTeam > Folder > Labels**. The **Properties** dialog box opens.
3. Select the **Labels** node. The **Labels** page of the **Properties** dialog box lists the labels currently attached to this folder.
4. Select the label to be detached from the folder.
5. Click **Detach**.
6. Optionally, if you are detaching a revision label, select one of the following options:
  - **Folder Only**
  - **Folder And Items Contained In Folder**
  - **Everything In Subtree Rooted At Folder**
7. Click **OK**.

 **Note:** When you detach a view label from a folder, the label is automatically detached from the items that the folder contains. It is also automatically detached from the child folders and their contents.

The folder from which the label is detached will disappear after a refresh.

## Detaching Labels from Items

If you decide not to include certain items in a view or revision label, you can detach the label from those items individually or as a group. Generally, labels are detached from files or folders.

 **Note:** You can attach and detach any labels from items in current view configurations, but you cannot see deleted items in those configurations. You can detach view labels from deleted items only if you roll back the view to a configuration based on the label you want to detach.


To detach a view or revision label from selected items

1. Select one or more items in the Server Explorer or in one of the Eclipse Explorers.
2. Right-click the selected items and choose **Labels > Detach** .  
The **Detach a Label** dialog box opens and displays all existing labels, identifying them as view or revision labels. By default, both **View Labels** and **Revision Labels** are checked.
3. Optionally, uncheck either **View Labels** or **Revision Labels** to limit the display list to a specific type of label.
4. Select a label from the list.
5. Click **OK**.

## Freezing or Unfreezing Labels

When a label is frozen, the label cannot be:

- Attached to any additional folders or items.
- Detached from any folder or items.
- Moved from one revision of a folder or item to another.

 **Tip:** You can identify a frozen label by a label icon containing a small snowflake displaying on a round, blue background. The icon displays in front of the label name in the list.

1. Choose **StarTeam > View > Labels** . The **Properties** dialog box opens.

2. Select the **Labels** node.
3. Do one of the following:
  - Click the **View** tab if the label to be frozen is a view label.
  - Click the **Revision** tab if the label to be frozen is a revision label.
4. Select the label from the list.
5. Click **Freeze** or **Unfreeze**.
6. Click **Close**.

## Promotion States

A promotion state is a state through which a product passes. For example, most software products go through a release or production cycle – that is, the product moves from developers to testers and back again, until it is ready to go to the marketplace. Promotion states provide a convenient mechanism for ensuring that the right files or other items are available to the right people at the right stage of this cycle. For example, if a software administrator creates Test and Release promotion states, files that are ready for testing can be assigned to the Test state and files that have been tested successfully can be assigned to the Release state.

The promotion state feature permits an administrator to create promotion states and associate a view label with each state. An administrator creates a new promotion state configuration which is the basis for a new view or a reconfigured view containing only items with a specified promotion state. Administrators can also set access rights for promotion states. The view labels assigned to a promotion state are usually also used as build labels, so that they can serve as properties in change requests.

The view label for a state can be changed whenever appropriate. It can also be promoted from one state to the succeeding state. For example, although testers may always be using files in the Test promotion state, the files may be from Build 07 in one week and from Build 08 in the next. Users usually configure the project view for their job assignment by promotion state instead of by view label. For example, testers would configure their view to the Test promotion state.

Many features of the application depend on calculations involving times and dates. In particular, labels, configurations, and promotion states are all governed by time and date calculations. If the clients and the Server are not kept synchronized, a number of operations (such as checkouts, file status displays, or label creation) may fail or produce inaccurate or unreliable results.

### Understanding Access Rights for Promotion States

Each view has its own set of promotion states. Access to these states is controlled by:

- The **Define Promotion Model** right which is set on the **View** node of the **Access Rights** dialog box for both projects and views. See “Granting Access Rights at the View Level”. A user with the **Define Promotion Model** right can do anything to the promotion model.
- Access rights that govern access to individual promotion states. These are **Generic Object Rights** and **Promotion State Specific Rights** which are set on the **Promotion State** node of the **Access Rights** dialog for both projects and views. They also appear on the access rights for individual promotion states.

The rights for an individual promotion state are checked at the state level; if necessary, the checking continues at the view level and eventually the project level. If a user is granted a given right at one level, there is no need to check the next.

- When a right is granted at the view level, it applies to all states in the view, unless access is denied at the state level.
- When a right is granted at the project level, it applies to all the states in all the views within the project, unless access is denied at the state or view levels.

## Example of Using Promotion States

Suppose a software company wants to use the following promotion states to correspond with its use of the application:

**Development** Developers work with the tip revisions of files. These files have no view label because they are constantly changing. Many companies do not use Development as a promotion state, because configuring a view to a promotion state, even when the view label for the state is `<current>`, makes the view read-only.

**White Box Test** Testers check both the source code and the compiled executable file for problems that need to be fixed. The source code will have a view label to ensure that the testers are looking at an unchanging set of files. The view label will be attached to a promotion state named White Box Test. (White box testing is testing with full knowledge of what is in the source code.)

The executable files are not stored in the application because they can be easily built from the source code. Testers install them from a `Builds` folder on the network. This folder has child folders named `Build 1`, `Build 2`, and so on.

Change requests are entered against the executable files only. Developers make repairs in the current source code, sometimes reviewing the files with the view label attached to the Black Box Test promotion state.

**Black Box Test** Testers install the executable file, just as they would with white box testing. However, they do not need to see the source code or use promotion states with it. (Black box testing is testing with no knowledge of what is in the source code.)

Change requests are entered against the executable files only. Developers make repairs in the current source code, sometimes reviewing the files with the view label attached to the Black Box Test promotion state.

**Alpha Test** End users of the software product being developed install the product executable files and test the product in their own environments.

Change requests are entered by the alpha coordinator and/or the users against the executable files only. Developers make repairs in the current source code, sometimes reviewing the files with the view label attached to the Alpha promotion state.

**Beta Test** Beta testing is similar to alpha testing, but the group of users is greatly expanded because the product is much more stable.

Change requests are entered by the beta coordinator and/or the users against the executable file only. Developers make repairs in the current source code, sometimes reviewing the files with the view label attached to the Beta promotion state.

**Release** The product is now sold in the marketplace. Users install the executable file and call product support. Product support enters change requests against the executable files only. Developers make repairs in the current source code, sometimes reviewing the files with the view label attached to the Release promotion state.

The fixes go into future product releases and service packs to releases already in the marketplace.

In this example, every time the source files are used to produce a build (set of executable files) for testing, a view label is applied to the files to identify them for future reference. It is convenient to use view labels such as **Build 1**, **Build 2**, and so on, so that it is clear which source code files were used to create which set of executable files.

Over time, the build or view label associated with a promotion state will change. For example, the Release state may initially be associated with `<current>`, rather than a view label, because no files are candidates for release and no appropriate view label has been created. When white box testers decide that the set of

files that they have examined is ready for black box testing, the view label associated with the White Box Test promotion state will be moved to the Black Box Test promotion state, and so on.

If promotion states are used, developers and testers who look at source code do not need to know that view label **Build 120** is currently being checked by white box testers, that the executable files for **Build 117** are currently undergoing black box testing, and other details.

## Promotion State Access Rights

Each view has its own set of promotion states. Access to these states is controlled by the “Define promotion model” right, which is available from the **View** node of the **Access Rights** dialog at the view and project levels. A user with the `Define promotion level` right can do anything to the promotion model, for example create and delete states, edit their properties, promote a label from one state to another. Promotion is a subset of editing properties. Anyone who can edit the properties of a state can also promote that state and reorder the states within the view.

Access rights that govern access to individual promotion states. These **Generic object rights** and **Promotion state specific rights** are available from the **Promotion State** node of the **Access Rights** dialog at the view and project levels. They also appear on the access rights for individual promotion states. The rights for an individual promotion state are checked at the state level; if necessary, the checking continues at the view level and eventually the project level. If a user is granted a given right at one level, there is no need to check the next.

When a right is granted at the view level, it applies to all states in the view, unless access is denied at the state level. When a right is granted at the project level, it applies to all the states in all the views within the project, unless access is denied at the state or view levels.

### Generic object rights

<b>Change object access rights</b>	Change the access rights for an individual promotion state. If you change this setting, be sure that you remain one of the users who can change access rights. This right is a generic object right. After creating a promotion state, you must exit and reenter the Promotion dialog if you want to set access rights for the newly created state.
------------------------------------	---

### Promotion State specific rights

<b>Modify label assignment</b>	Change the label assigned to an individual state either by clicking the Promote button or editing the label property. No other properties for the state can be edited unless the user also has the Define promotion model access right from the View node. This right is a promotion state specific right.
--------------------------------	--

## Configuring Promotion States

When creating promotion states, many administrators assign `<current>` to the initial promotion state instead of a view label, because that state always uses the tip revisions. They also often assign `<current>` to later promotion states for which no view labels currently exist. These states may receive a view label later, when the files associated with a view label meet the criteria required by the state. Alternatively, a view label may be promoted from the preceding state to the label-less state.

### Creating a New Promotion State



**Note:** You can create promotion states only if you have the required access rights, which are found at the project or view level.

1. Choose **StarTeam > View > Promotion** from the main menu. The **Properties** dialog box opens.
2. Select the **Promotion** node. The **Promotion** page of the **Properties** dialog box displays the states currently created for this view. The final state displays at the top of the list.



3. Click **Add** to open the **Promotion State** dialog box.
4. Type the **Name** and **Description** of the promotion state.
5. Assign a view label to this state by selecting it from the **View Label** drop-down list. The labels are listed in reverse chronological order, based on the time at which they were created. You can change the label when appropriate by using this dialog box or assign it to the next state by promoting it.
6. Click **OK** to close the **Promotion State** dialog box, and **OK** again to close the **Promotion** dialog box.

### Editing or Deleting a Promotion State

1. Choose **StarTeam > View > Promotion** from the main menu. The **Properties** dialog box opens.
2. Select the **Promotion** node. The **Promotion** page of the **Properties** dialog box displays the states currently created for this view. The final state displays at the top of the list.
3. Click **Edit**. The **Promotion State** dialog box opens.
4. Modify the **Name**, **Description**, or **View Label**.
5. Click **OK** to close the **Promotion State** dialog box, and **OK** again to close the **Promotion** dialog box.

## Filters and Queries

A filter is a named arrangement of data that consists of a set of fields (used as column headers), sorting and grouping information, and (usually) a query. Once a filter has been created, it can be used in every project that has the same server configuration. Queries limit the items displayed in a view and are usually used when defining a filter.

In the **Change Request**, **Requirement**, **Task**, **Topic**, and **Audit** views, a list box automatically displays a list of predefined filters. For example, the **Change Request** view provides a *Priority* filter that when selected, automatically sorts and groups change requests by their specified priority. You can create and edit filters using **Filters** found in the list box toolbar menu for the **Change Request**, **Requirement**, **Task**, and **Topic** views.

You can define and edit additional filters and queries using **Filters** and **Queries** found in the corresponding list box toolbar menu for the **Change Request**, **Requirement**, **Task**, and **Topic** views. Once you have created and publicly saved your filters/queries for a server configuration, the list displays these filters.

## Filtering Data

A filter lets you view data in a particular arrangement that consists of a set of fields (used as column headers), sorting and grouping information, and (usually) a query. Once a filter has been created, you can use it in every project that has the same server configuration.

### Applying Predefined Filters

Existing public filters can be used on all projects in the same server configuration by any team members who have the appropriate access rights. Private filters can be used only by you.

1. Select a folder in the Server Explorer or one of the Eclipse Explorers.
2. Open one of the following views: **Change Request**, **Requirement**, **Task**, **Topic**, or **Audit Log**.
3. Select a filter from the **Filters** list.
4. Click **OK**.

### Copying a Filter

1. Select a folder in the Server Explorer or one of the Eclipse Explorers.
2. Open one of the following views: **Change Request**, **Requirement**, **Task**, **Topic**, or **Audit Log**.

3. Choose **Filters > Filters**. The **Filters** dialog box appears.
4. Select a filter from the **Filters** list.
5. Click **Save As**. The **Save Filter As** dialog appears.
6. Type a name for this filter in the **Filter Name** field.
7. Select or clear the **Public** check box. If the filter includes a query, the status of the new filter must be the same as the status of the original filter.
8. Click **OK**.
9. Do one of the following:
  - Click **Select** to apply the new filter to the upper pane.
  - Click **Close** to exit without applying the new filter.

## Deleting Filters

If desired, you can delete filters that you no longer use.

1. Select a folder in the Server Explorer or one of the Eclipse Explorers.
2. Open one of the following views: **Change Request**, **Requirement**, **Task**, **Topic**, or **Audit Log**.
3. Choose **Filters > Filters**. The **Filters** dialog box appears.
4. Select a filter from the **Filters** list.
5. Click **Delete**.
6. When a message box asks you to confirm your deletion, click **OK**. This action returns you to the **Filter** dialog box.
7. Click **Close**.

## Editing Filters

You edit filters by changing their fields, sort orders, or queries.

1. Select a folder in the Server Explorer or one of the Eclipse Explorers.
2. Open one of the following views: **Change Request**, **Requirement**, **Task**, **Topic**, or **Audit Log**.
3. Choose **Filters > Filters**. The **Filters** dialog box appears.
4. Select a filter from the **Filters** list.
5. Edit any of the following:
  - Fields button** Select the column header fields.
  - Sort, Group button** Sort and group items in up to four fields in ascending or descending order.
  - Query button** Limit the items that appear in the upper pane to those that match the query.
  - Context button (for files only)** Specify the files that will be affected by the filter. Clicking this button opens the **Set Filter Type** dialog. On this dialog box, apply the filter to one of the following by selecting an option button: **Items in the view** is equivalent to applying both your filter and the **Files in view** filter. **Items not in the view** is equivalent to applying both your filter and the **Files not in view** filter. **All items not excluded from the view** is equivalent to applying both your filter and the filter.
6. Click **Save As**. The **Save Filter As** dialog box appears. Do *not* change the name of the filter.
7. Click **OK** to return to the **Filters** dialog box.
8. Do one of the following:
  - Click **Select** to apply the edited filter to the upper pane.
  - Click **Close** to exit without applying the edited filter.

## Resetting Filters

In StarTeam, you can apply a filter, then rearrange the data on the upper pane or apply a new query. Doing this places an asterisk in front of the filter's name, showing that it has been changed. After looking at the new data, you can then reset the filter as it was originally defined on the server, which removes the asterisk.

1. Select a folder in the Server Explorer or one of the Eclipse Explorers.
2. Open one of the following views: **Change Request**, **Requirement**, **Task**, **Topic**, or **Audit Log**.
3. Choose **Filters > Filters**. The **Filters** dialog box appears.
4. Do one of the following:
  - Right-click a column header on the upper pane, then select **Reset Current Settings**.
  - Choose **Filters > Reset Current Settings** from the component or context menu.

The system asks: `Reset filter: <Filter>?`

5. When the system asks: `Reset filter: <Filter>?`, click **OK**. This action resets the filter and removes the asterisk.

## Change Request Filters

### Change Request > Filters > Filters

Filtering allows you to limit the types and numbers of items that appear in the upper pane. The list of filters depends on your selection from the folder tree and whether the **All Descendants** button is selected from the toolbar or component menu.

StarTeam provides a set of predefined filters that are intended as starting points for you to create your own custom filters. Use the **Filter** list on the toolbar to view and apply predefined file filters.

<b>&lt;Show All&gt;</b>	Displays all items.
<b>By Status and Responsibility</b>	Groups change requests based on their statuses and the users who are currently responsible for processing the requests.
<b>Not a Priority</b>	Displays only the change requests that are not a priority.
<b>Priority</b>	Displays only the change requests that are a priority.
<b>Show Unread Changes</b>	Displays only the change requests that you have not read (or not read since they were modified).
<b>Status = Closed</b>	Displays only the change requests that are closed.
<b>Status = Deferred</b>	Displays only the change requests that are postponed.
<b>Status = Open</b>	Displays only the change requests that are open and in progress.
<b>Status = Resolved</b>	Displays all the change requests that have one of the following statuses: <b>As Designed</b> , <b>Cannot Reproduce</b> , <b>Documented</b> , <b>Fixed</b> , or <b>Is Duplicate</b> .
<b>Status = Verified</b>	Displays all the change requests that have one of the following statuses: <b>Verified As Designed</b> , <b>Verified Cannot Reproduce</b> , <b>Verified Documented</b> , <b>Verified Fixed</b> , or <b>Verified Is Duplicate</b> .
<b>Type = Defect</b>	Displays only the change requests that have the type <b>Defect</b> .
<b>Type = Suggestion</b>	Displays only the change requests that have the type <b>Suggestion</b> .

## Requirement Filters

### Requirement > Filters > Filters

Filtering allows you to limit the types and numbers of items that appear in the upper pane. The list of filters depends on your selection from the folder tree and whether the **All Descendants** button is selected from the toolbar or component menu.

StarTeam provides a set of predefined filters that are intended as starting points for you to create your own custom filters. Use the **Filter** list on the toolbar to view and apply predefined file filters.

<b>&lt;Show All&gt;</b>	Displays all items.
<b>&lt;Flagged Items&gt;</b>	Lists only requirements that have been flagged.
<b>Grouped by Creator</b>	Displays groups of requirements, one group for each user who has created requirements.
<b>Grouped by Status</b>	Displays groups of requirements, one group for each existing status.
<b>I Am Responsible</b>	Displays only the requirements for which you are responsible.

## Task Filters

### Task > Filters > Filters

Filtering allows you to limit the types and numbers of items that appear in the upper pane. The list of filters depends on your selection from the folder tree and whether the **All Descendants** button is selected from the toolbar or component menu.

StarTeam provides a set of predefined filters that are intended as starting points for you to create your own custom filters. Use the **Filter** list on the toolbar to view and apply predefined file filters.

Examples of custom task filters that you might create include:

- `Responsibility Equals <user name>`, which identifies only the tasks for which a specific person is responsible.
- `Percent Complete < 100`, which identifies unfinished tasks.

<b>&lt;Show All&gt;</b>	Displays all items.
-------------------------	---------------------

## Topic Filters

### Topic > Filters > Filters

Filtering allows you to limit the types and numbers of items that appear in the upper pane. The list of filters depends on your selection from the folder tree and whether the **All Descendants** button is selected from the toolbar or component menu.

StarTeam provides a set of predefined filters that are intended as starting points for you to create your own custom filters. Use the **Filter** list on the toolbar to view and apply predefined file filters.

<b>&lt;I Am Recipient&gt;</b>	Identifies all the topics that name you as a recipient.
<b>By Creator</b>	Groups the topics by their original authors.
<b>Show Active</b>	Identifies all topics and responses that have <code>Active</code> status.
<b>&lt;Show All&gt;</b>	Displays all items.


## Audit Filters

### Audit > Filters > Filters

Filtering allows you to limit the types and numbers of items that appear in the upper pane. The list of filters depends on your selection from the folder tree and whether the **All Descendants** button is selected from the toolbar or component menu.

StarTeam provides a set of predefined filters that are intended as starting points for you to create your own custom filters. Use the **Filter** list on the toolbar to view and apply predefined file filters.

<b>&lt;By Class and Event&gt;</b>	Displays audit entries sorted by their value in the <b>Class Name 1</b> field (type of item) and <b>Event</b> (type of action) field.
<b>By Transaction and Event</b>	Groups audit log entries by descending <b>Transaction ID</b> and then by <b>Event</b> type. This filter provides a reverse-chronological list of updates in the view by transaction.
<b>Events</b>	Groups audit log entries by <b>Event</b> type, then by <b>Target 1 Class ID</b> , and then by <b>Created Time</b> .
<b>Show All</b>	Displays all entries (the default).

 **Note:** You can limit the number of audit log entries displayed by creating a query that selects audit log entries by specific property values.


## Queries

You can use a query to limit the items that display. Each query is performed on all items in the StarTeam folder and component you have selected. The fields included in the query do not have to display. Once a query has been created, it can be used in every project in the same server configuration.

You can use the following components: Change Request, Requirement, Task, Topic, or Audit Log views.

StarTeam queries have the following attributes:

- A unique name that easily identifies the query. Query names are not case-sensitive.
- Public or private status. Anyone with appropriate access rights can use public queries, while private queries are available only to your user ID. Once a query has been saved with a specific status, its status cannot be changed. However, you can copy a query and change the state of the new query.
- A logical expression appropriate for items of a particular type. These expressions include one or more conditions. A condition consists of a field (not necessarily a current column header), a relational operator, and a value to be compared to the value of the field. For example, a condition used to locate change requests might be: `Responsibility Equals Rhonda Thurman`. More complex queries include two or more conditions bound together by logical operators: `AND`, `OR`, and `NOT`. For example, to locate all the change requests for which Rhonda Thurman is responsible that also have a high severity, use: `Responsibility Equals Rhonda Thurman AND Severity Equals High`.



 **Note:** If you are creating a complex query, and the first logical operator in your query should be `OR`, select the `AND` logical operator in the query tree. Then click the **AND->OR->NOT** button. This changes an `AND` to an `OR`. Similarly, one more click changes the `OR` to a `NOT`. Keep toggling the button until the operator that appears is the one you want to use. It is best to use the condition or logical operation that will result in the fewest matches as the first condition or logical operation.


## Applying Queries

1. Select a folder in the Server Explorer or one of the Eclipse Explorers.
2. Open one of the following views: **Change Request**, **Requirement**, **Task**, **Topic**, or **Audit Log**.
3. Choose **Filters > Queries**. The **Queries** dialog box opens.
4. Select a query from the list.
5. Click **Select** to apply the query to the items shown in the view. This action changes the contents of the view by displaying only those items that meet the specifications of the query.

## Copying Queries


StarTeam allows you to create new queries quickly by copying an existing query and editing it. Using this feature saves time because you do not have to recreate the query conditions.

1. Select a folder in the Server Explorer or one of the Eclipse Explorers.
2. Open one of the following views: **Change Request**, **Requirement**, **Task**, **Topic**, or **Audit Log**.
3. Choose **Filters > Queries** . The **Queries** dialog box opens.
4. Select a query from the list.
5. Click **Copy**. The **Copy Query** dialog box opens.
  -  **Tip:** Public queries have a multi-user icon to the left of the query name. Private queries have a single-user icon.
6. Type a name for the query in the **Query Name** field.
7. Select the **Public** check box to add this query to the project (and the server configuration), allowing anyone with the appropriate access rights to use it. If you do not check the **Public** check box, the query will be private, that is, available only to your user ID.
8. Click **OK**. The new query displays in the **Queries** dialog box.
9. To change the conditions in the query, select it from the **Queries** dialog box and click **Edit**. The **Edit Query** dialog box opens.
10. Edit the appropriate nodes of the tree.
11. Click **Save**. The **Queries** dialog box opens.
  -  **Note:** If you do not have the access rights to create a public query for this project, you can create a private query.
12. Click **Close**.

 **Note:** If this is a public query, you might want to set access rights for it.

## Creating Queries

You can write simple queries that have only one condition, or complex queries that use several conditions and one or more logical operators.

1. Select a folder in the Server Explorer or one of the Eclipse Explorers.
2. Open one of the following views: **Change Request**, **Requirement**, **Task**, **Topic**, or **Audit Log**.
3. Choose **Filters > Queries** . The **Queries** dialog box opens.
4. Click **New**. The **New Query** dialog box opens.
5. Type a name for the query in the **Query Name** field.
6. Select the **Public** check box to add this query to the project (and the server configuration), allowing anyone with the appropriate access rights to use it. If you do not check the **Public** check box, the query will be private, that is, available only to your user ID.
7. Select a **Field** and **Operator**, type or select a **Value**, and click **Add** to place this condition in the **Query** tree.
  - By default, the **Query** tree contains the **AND** operator as the root of the tree, which you cannot delete. If there is only one condition, StarTeam ignores the logical operator.
8. Click **View as Text** to view the query in text format. Notice that the default logical **AND** operator was not included in your query. Click **OK**.
9. Optionally, click one of the following **Logical Node** buttons to create a new **Query** tree node: **AND**, **OR**, or **NOT**.
  -  **Tip:** You can change an existing operator in a condition by toggling the **AND->OR->NOT** button. Keep clicking the button until the operator that appears is the one you want to use.
10. Select the fields for this new condition and click **Add**.
11. Add any other conditions, then click **Save**.

The **Queries** dialog box now contains your new query enabling you to select it for querying data.



**Note:** If this is a public query, you might want to set access rights for it.



**Tip:** When creating a query condition, it is best to use the condition or logical operation that will result in the fewest matches as the first condition or logical operation.

## Creating "Me" Queries

StarTeam has the capability of creating "Me" queries that allow a query to be set up which is evaluated against the currently logged in user ("Me"), rather than having to specify a specific username at the time of query creation.

1. Select a folder in the Server Explorer or one of the Eclipse Explorers.
2. Open one of the following views: **Change Request**, **Requirement**, **Task**, **Topic**, or **Audit Log**.
3. Choose **Filters > Queries** . The **Queries** dialog box opens.
4. Click **New**. The **New Query** dialog box opens.
5. Type a name for the query in the **Query Name** field.
6. Select the **Public** check box to add this query to the project (and the server configuration), allowing anyone with the appropriate access rights to use it. If you do not check the **Public** check box, the query will be private, that is, available only to your user ID.
7. Select **Created By** for the **Field** in the **Condition Node** area. Select **Equals** for the **Operator**, and select **Me** in the **Value** list. Click **Add** to place this condition in the **Query** tree.

By default, the **Query** tree contains the **AND** operator as the root of the tree, which you cannot delete. If there is only one condition, StarTeam ignores the logical operator.

8. Click **Save**. The **Queries** dialog box now contains your new query enabling you to select it for querying data.

## Deleting Queries

You can delete queries that you are sure you no longer use.



**Note:**

- You must have the appropriate access rights to delete a public query.
- You cannot delete a query that is referenced by a filter.

1. Select a folder in the Server Explorer or one of the Eclipse Explorers.
2. Open one of the following views: **Change Request**, **Requirement**, **Task**, **Topic**, or **Audit Log**.
3. Choose **Filters > Queries** . The **Queries** dialog box opens.
4. Select a query.
5. Click **Delete**.
6. Click **OK**. The query is removed.
7. Click **Close**.

## Editing Queries

To display a useful set of data, you might need to edit or add to a query.

1. Select a folder in the Server Explorer or one of the Eclipse Explorers.
2. Open one of the following views: **Change Request**, **Requirement**, **Task**, **Topic**, or **Audit Log**.
3. Choose **Filters > Queries** . The **Queries** dialog box opens.
4. Select a query.
5. Click **Edit**. The **Edit Query** dialog box opens.
6. Edit the appropriate nodes of the tree.

7. Click **Save**. The **Queries** dialog box opens listing the edited query.
8. Click **Close**.

## Predefined Queries

Initially, the **Queries** dialog box lists default queries that are predefined for the current component. You can apply a default query, edit a default query, create a new query, or delete a default query.



**Note:** Default queries that do not appear in your list might have been deleted after installation.

<b>File</b>	Files to Check In, Files to Check Out, Flagged Items.
<b>Change Request</b>	Flagged Items, Not a Priority, Priority, Status=Closed, Status=Open, Status=Resolved, Status=Verified, Type=Defect, Type=Suggestion, Unread Changes.
<b>Requirement</b>	Flagged Items, I Am Responsible.
<b>Task</b>	Flagged Items.
<b>Topic</b>	Flagged Items, I Am Recipient, Show Active.
<b>Folder</b>	Folders Not In View.
<b>Audit</b>	None.

## Relational Operators Used in Queries

The relational operators that you can use to define conditions in a query vary according to the type of field:

- Text fields
- Boolean, enumerated type, and numeric fields
- Date/time fields

### Relational Operators Used on Text Fields

The relational operators that can be used on text fields are:

- Equals
- Is Not
- Contains (ignore case)
- Contains (match case)
- Starts with (ignore case)
- Starts with (match case)
- Ends with (ignore case)
- Ends with (match case)

### Relational Operators Used on Boolean, Enumerated Type, and Numeric fields

The relational operators that can be used on Boolean, enumerated type, and numeric fields are:

- Less Than
- Same or Less
- Equals
- Same or Greater
- Greater Than
- Is Not



## Relational Operators Used on Date/Time Fields

The relational operators that can be used on date/time fields are listed below.

### Comparing both date and time parts of date/time fields

- Before
- On or Before
- On
- On or After
- After
- Not On

### Comparing only the date part of date/time fields

- Before Date
- On or Before Date
- On Date
- On or After Date
- After Date

### Matching all dates starting with the date that was the specified a number of days or weeks ago

- Last (n) Days
- Last (n) Weeks

### Matching all the dates prior to and including the date that was the specified number of days or weeks ago

- Older Than (n) Days
- Older Than (n) Weeks



**Note:** In date fields, StarTeam treats blanks as zeroes. That means that “no date” is less than any specific date. For example, if you write a query that searches for change requests that were closed prior to some specific date, all the change requests with no date in the **Closed On** field are included in the results, even though they have not been closed yet. It is easy to eliminate the change requests that contain blanks in the **Closed On** field from such a query. You simply AND the condition that searches for change requests closed on or before a specific date with another condition that searches for change requests closed after the date zero.

## New Query Options

Click **New** to define a new query in the **New Query** dialog box.

<b>Query</b>	Displays the query definition parameters.
<b>Name</b>	Enter a unique name for the new query.
<b>Public Check Box</b>	Check to give the query public status. Public queries can be used by anyone with the appropriate access rights, while private queries are available only to your user ID. Once a query has been saved with a specific status, its status cannot be changed. However, you can copy a query and change the state of the new query.
<b>Query List</b>	Displays the query definition. By default, the AND condition appears as a starting logical condition.

**Logical node** Use to choose the logical operator for each condition.

<b>AND</b>	Click to add an AND condition to the query.
<b>OR</b>	Click to add an OR condition to the query.
<b>NOT</b>	Click to add a NOT condition to the query.
<b>AND-&gt;OR-&gt;NOT</b>	Click to toggle the default AND condition to an OR or NOT condition.

	<b>Remove</b>	Click to delete the currently selected condition. You must confirm the deletion.
<b>Condition Node</b>		Use to define the conditions for the logical operator.
	<b>Field</b>	Lists all the fields available for this component.
	<b>Operator</b>	Lists all of the operators that can be specified for the selected field.
	<b>Value</b>	Use to specify a value for this field and condition.
	<b>Show advanced fields check box</b>	Select to display all possible fields including the advanced fields.
	<b>Show deleted users check box (optional)</b>	For components with user fields only (such as the Change Requests component), select to show deleted users in the query results.
	<b>Alphabetical check box (optional)</b>	For enumerated fields only (fields that have specified values), select to alphabetize the query results rather than list them in the order in which they appear in the enumeration list.
	<b>Add</b>	Click to add the condition to the query definition.
	<b>Modify</b>	Click to modify the selected query condition.
	<b>Delete</b>	Click to delete the selected query condition.
<b>View as Text button</b>		Displays the current query definition in a field.
<b>Save button</b>		Saves the query.
<b>Cancel button</b>		Cancel the query definition.

## Queries Options

Use the **Queries** dialog box to view and apply the currently defined queries.

<b>Queries list</b>	Lists all existing queries for this component. Note that the multi-person icon left of the query name indicates a public query; a single-person icon indicates a private query.
<b>Close button</b>	Closes the <b>Queries</b> dialog box.
<b>Select button</b>	Applies the selected query to the data listed in the selected component.
<b>Clear Query button</b>	Clears the current query and displays all data.
<b>New button</b>	Opens the <b>New Query</b> dialog box for you to define a new query.
<b>Edit button</b>	Opens the selected query definition in the <b>Edit Query</b> dialog box for you to edit.
<b>Copy button</b>	Opens the <b>Copy Query</b> dialog box for you to enter a name for your new query.
<b>Delete button</b>	Deletes the selected query definition.
<b>Access Rights button</b>	Allows you to assign access rights to a query.

# Terminology

This section defines some of the more common terms used in StarTeam that are used throughout the documentation.

Access rights	A security feature. The rights granted (or denied, in exceptional cases) to users or groups that allow team members to see, modify, create, and delete items.
Adding files	Process of placing files under version control by adding them to a project view.
Administrator	A functional role in the operation of the application. Administrators install, configure/set up, create and/or import projects, manage server configurations, users and groups, and so on.
Alternate working folder	Allows you to store project files on your workstation at a specific location you have selected. Using an alternate working folder for the root of a view or a branch in an application folder hierarchy can also alter the path of the working folders used for child folders.
Application file	A file under version control; therefore, a file that is in an application project.
Application folder	A folder in the application folder hierarchy. Application folders help organize the project view into discrete understandable parts. Each folder has a working folder that corresponds to it and exists on your hard drive.
Application folder hierarchy	The hierarchy of folders as it appears in the Server Explorer. When you map a view and use the folder hierarchy, the folder structure on the virtual disk matches the folder structure you see in the application. This may or may not match the working folder structure on your workstation.
Archive	File or group of files that makes it possible to recreate past revisions of a file under version control. An archive can also be, as in the application, the folder that stores such files.
Audit entry	Record of an action performed on an application project that appears in the audit log.
Audit log	Chronological record kept by the application that shows all actions performed on folders and items, such as files and change requests.
Author	User that created a revision.
Blank branching view	An empty branching view. It has no correlation to the parent view. Also called non-derived view.
Branch	An independent item derived from a corresponding item in a parent view. In the case of a text file, the branched item can later be merged with the file from which it originated. This term also refers to a branch of a tree, such as the folder hierarchy or a topic tree.
Branch revision	A number is assigned by the application to a revision of an item. It indicates how many consecutive revisions have been made in a view; it branches when the item is branched into a new view.

Branching view	A view that may or may not be derived from a parent view. When not derived from a parent view, it is a blank branching view. Branching views always permit branching. If they float and have the <b>Branch on change</b> option set, they are updated by the parent view on a file-by-file basis until the file changes in the branching view. If they float and do not have the <b>Branch on change</b> option set, updates are sent to the parent view until the <b>Branch on change</b> option is set. If they are based on a label, a promotion state, or a moment in time, they are read-only, unless the <b>Branch on change</b> option is set.
Build	The process of compiling, assembling, and linking all project files in proper sequence to produce a software product. Also an event in the life cycle of a product chosen to represent a quantifiable step in progress for a project.
Build label	Label for a particular product build; in the application, a view label may serve as a build label.
Change request	Item that reports a fault or error in a product or suggests an enhancement.
Check in	Operation that stores a new file revision in the repository. The person who checks the file in can keep it locked or release the file to others by unlocking it.
Check out	Operation that retrieves a revision of a file from the application Server and places it in a working folder for the user. A user can check out a file with or without the intention of altering that file. Locking the file marks the beginning of a revision.
Compression	Data transferred between your workstation and the server can be compressed to reduce the amount of traffic on the network. However, the time to compress and decompress the data is added to the transfer time.
Configuration	A relative arrangement of parts or elements. An application configuration isolates a view, folder, or item to a particular revision. For example, you can configure a view to be current or based on a view label, a promotion state, or a selected date and time.
Container	Term indicating the ability to contain other types of items. For example a project is a container for views, folders, and items. Views and folders are also containers.
Current	File status. Content of the file in the working folder is the same as the content of the tip revision of the file.
Delta storage	Method of computing differences between progressive revisions of a file.
Exclusive lock	An exclusive lock indicates to others that you intend to make changes on an item. For a file, the icon others see is yellow padlock, while you see an icon with a small yellow key and the head and shoulders of a person. Other people cannot check in an item that you have exclusively locked.
File compression	Technique for reducing the size of a file by removing redundant information. Most disk files contain repetitions of common sequences of characters. Compression algorithms remove the additional occurrences of these sequences and save the information that permits their restoration. Selecting file compression may reduce vault space requirements, as well as improve performance.

Filter	Criteria used to select a few items from among many. The <b>Filter</b> list box allows you to display only the items that are of interest. Applying a filter may also control what columns are displayed, what columns are sorted, and how the values are grouped in the sorted columns.
Fixed change request	Designation indicating that a change request has been resolved.
Float	For views, a floating view is one that stays current with its parent view. In other words, updates to the parent view are sent to the child view. If the child is not read-only, updates to the child view also go to the parent.
Folder hierarchy	Hierarchical display of an application view and its associated folders. The folder hierarchy is always displayed in the Server Explorer.
Frozen	An item is said to be frozen (and, therefore, read-only) if it is based on the state of the corresponding item in the parent view at a specific moment in time and cannot be branched. An item is also frozen if you reconfigure it to a specific label, promotion state, or time in its past.
History list	List of revisions for the file selected from the Server Explorer or from one of the Eclipse Explorers. The application displays the History list when you open the History view.
Item	The application object or element. Items include files, change requests, requirements, tasks, topics, responses, and audit entries. A folder is also treated as an item, even though it is a container for items.
Keyword	Reserved words beginning and ending with a dollar sign (\$). When used in a text file, the application replaces these words with the data that they represent. For example \$Author\$ is replaced by the name of the user who checked in the file.
Keyword expansion	A technique used to insert information in a text file. Keywords are replaced by the data they represent.
Label	In version control, the act of attaching a view or revision label (name) to one or more folders and/or items.
Labeled configuration	The basis for a new or reconfigured view. The view contains only the items with the label you specify.
Link	Method of connecting any two folders, items, or a folder and an item. If you select an item, the client provides a Link view that shows all other items to which it is linked.
Lock	File locking is a technique used to inform others that you are revising a file (exclusive lock) or considering its revision (non-exclusive lock). Checking files out or in does not imply automatic locking or unlocking in the application, as these operations can be performed manually. Locks can also be overridden and broken.
MAPI	Acronym for Mail Application Programming Interface, a programming interface that permits an application to send and receive electronic mail via the Microsoft Mail messaging system. The application uses SMTP and not MAPI.
Merge	File status that indicates that the working file is not based on the tip revision of the file. When you check this file in or out, the application asks if you want to merge it with the tip revision; or

	<p>Process of combining a working file with the tip revision of that file or combining a branched file with the original file from which it was branched. This operation involves a three-way comparison, in which both files are compared with the file revision that is their most recent common ancestor. The combined file can be inspected or revised by the user before it is checked in.</p>
Milestone	<p>An event in the life cycle of a product that represents a significant step in its progress, for example, its final release. In the application, when you reach a milestone, you can apply a view label (usually a build label) to indicate that the milestone has been reached.</p> <p>In Microsoft Project, a milestone is a type of task that represents a significant landmark, development, or turning point in the life of a scheduled project. You usually define a milestone by entering a task name and assigning it a duration of zero. Milestones typically signal that the work has started or is completed and do not represent the act of doing of the work.</p>
Missing	<p>File status indicating that the file is not in your working folder. You may want to check the file out.</p>
Modified	<p>File status indicating that the working file has been altered and is based on the tip revision of this file. You may want to check this file in.</p>
Non-exclusive lock	<p>A lock that notifies others that you are considering changing the file. When you have a file locked non-exclusively, others can check the file in.</p>
Not In View	<p>File status that indicates the file is in the working folder, but not in the application view. You may want to add this file to the view.</p>
Notification of events	<p>If this feature is enabled, the application notifies that you have become responsible for a change request, that a requirement or task for which you are responsible has changed, or that a topic for which you are a recipient has changed. The notification appears as an appropriate icon on the status bar.</p>
Object	<p>Generic term used in object-oriented programming and elsewhere to indicate something upon which operations can be performed.</p>
Out Of Date	<p>File status indicating that your working file is a copy of an old revision of the file. If you need the current revision, check it out.</p>
Preferences	<p>User-selectable choices for the behavior of the application on a specific workstation.</p>
Pooled connections	<p>Concurrent connections controlled internally by the server that the server opens to the database engine. The range is usually database-specific and may also depend upon the number of database licenses.</p>
Project	<p>A project is a set of related folders and their contents that usually represents a single product under application version control. Large, complex projects have many folders and files that are worked on by many team members. A project is the collection and organization of all these files and folders. A project might contain the files that comprise a software program, a technical publication, a legal case, a financial forecast, a building, an aircraft, or anything involving numerous files, each of which may undergo many revisions as the job progresses.</p>

Promotion state	A state through which a product passes. For example, a software application that goes through a development, test, and release cycle could use the promotion states Development, Test, and Release. In the application, each promotion state has a view label assigned to it.
Promotion state configuration	Basis for a new or reconfigured view. The view contain only items with the promotion state you specify.
Properties	Attributes stored for each item and each revision of an item under version control.
Protocol	Set of rules governing how something is done.
Query	Criteria used to select a few items from among many. You can apply a query to items in the view to display only the items of interest.
Read-only reference view	A reference view is a view that either cannot be changed at all or changes only when its parent view changes. If it floats, it is updated as its parent changes. If it is based on a label and the items with the specified label change in the parent view, the read-only reference view reflects that fact. If it is based on a specific date and time, it is frozen, to show what the parent view looked like at that point in time and cannot be rolled back.
Reconfiguring	A view, folder, or item can be reconfigured to a point in the past, as defined by a label, promotion state, or a point in time.
Reference count	List of the items that reference another item. For example, a file may be shared by two projects on the same server and, therefore, have two references to it.
Reference view	A view derived from a parent view. It generally uses a different folder as a root folder and the same working folders as those of its parent. If it floats, it receives updates as the parent view changes. If it floats and is not read-only, it sends updates to the parent view as it changes. If the reference view is based a specific label or date and time within the parent view, it is frozen at that moment in time and is read-only.
Repository	The database and hive associated with a particular server configuration. Each hive contains an Archives and a Cache folder for file storage.
Repository customization	The application feature that allows you to modify values of existing enumerated fields and create customized fields for items to represent information specific to your working environment. Your license determines whether you have this feature.
Requirement component	The application component that allows users to create, track, and complete requirements related to the project.
Response	Replies to a topic that, along with the topic, form a hierarchical structure called a topic tree.
Revision	Set of changes to an item. As an item is revised, each set of changes is saved as a revision.
Revision label	Method of identifying a revision of an item or a set of revisions by name; used primarily for files. For example, when you check in a group of files that may need to be checked out together, you can give them a revision label.

Revision number	The revision number is an identification number assigned by the application to a revision of an item. It indicates how many consecutive revisions have been made since the item was originally created.
Root folder	The top folder in an application folder hierarchy. It has the same name as the view.
Secondary sort	Sorting items by group in a list that is already sorted (primary sort). For example a File list might be sorted by extension, then sorted by name within groups of the same extension.
Server	A computer or system that provides services to clients and is running the StarTeam Server software. The clients may be other computers. The server controls the repository, which is a storage place for file revision archives, and a database that contains information about files, such as their descriptions, the number of revisions, and so on.
Server configuration	Contains the repository and option settings selected when you set up the application server. For example, the administrator may want projects to use encryption and compression, so the server configuration specifies these features. The application items, such as folders and files, can be shared provided their projects use the same server configuration. You can start a server with any one of several server configurations, but that configuration controls what projects, views, and so on that you can access during that session.
Shortcut	A file that starts another application, often with a specific document or set of data. The shortcut is usually stored as an icon on your desktop.
SMTP	Simple Mail Transfer Protocol, commonly used for Internet and UNIX mail systems. It usually uses port 25. Sending items via email in the application requires SMTP.
Sort	To place items in ascending or descending order in the view, based on the value in one column. Depending on the values in the column, values are sorted numerically, alphanumerically, or by an internal order or key. Click the column header to change the sort order from ascending to descending or vice versa.
Status	File status is the relationship between the working file and the tip revision in the repository. The file statuses are: In View, Not in View, Missing, Current, Merge, Modified, Out of Date, and Unknown.
Storage method	Revisions stored in the archive (or vault) for a specific file can be changed from one type of storage and compression to another. There are two types of storage: forward delta storage, which is recommended for text files, and full revision storage, which is recommended for binary files and text files that change massively from revision to revision.
Task component	Users can create, track, and resolve tasks related to their projects with this component if your company has licensed Enterprise Advantage or Enterprise. The component also interoperates with Microsoft Project.
TCP/IP	A protocol for communication between computers used by the Internet; acronym for Transmission Control Protocol/Internet Protocol.
Test command	Command that you enter so that the application can perform an automated procedure to test a defect.



Text file or ASCII file	File that contains only printable text characters, spaces, carriage returns, and sometimes tabs and an end-of-file marker, without formatting codes. The application identifies as text files any files that contain no null characters. All other files are binary.
Time stamp	Information maintained by the application about files and revisions. For file revisions, the time stamp is the date and time that the file was checked into the application. For files, it is the date and time for the working file.
Tip revision	Most recent revision to an item. The tip revision is based on your view configuration. For example, if your view is configured to a particular label, the tip revision is the revision with that label. If your view is configured to a particular date and time, the tip revision is the one that was checked in just prior to that point in time.
Title bar	Bar that shows the name of the application and, if it is maximized, of the selected child window. For a view, the child window name includes both the project name and the view name, followed by the current working folder for the root folder.
Toolbar	Bar that contains buttons or icons, each of which performs a variety of functions. Usually located at the top of the screen, but can be dragged to any edge of the window or float in a separate window that can move anywhere on the screen.
Topic	First message on a particular subject attached to a folder in the application folder hierarchy. After this message has been submitted by a team member, others may respond to it, creating a topic tree.
Topic tree	Topic tree represents a threaded conversation of the topic with its responses and the responses to those responses.
Unknown	File status indicating that a file in the working folder has the same name as a file in the view, but that the file has not been checked out from the repository. You may have copied the file from another location.  Use the <code>Update Status</code> command to determine the correct status.
Unlock	The process of releasing a locked file, indicating to others that you no longer wish to change the file.
User	An individual given access to an application server configuration and the projects it manages.
Vault	Folder in which revisions of the files that are under version control are stored.
Vault file	File in the vault that stores file revisions.
Version control	Version control is the process of storing and tracking changes (revisions) to one or more files. The main advantage of using an automated version control system is fast, easy recall of previous revisions. The application also tracks revisions of other items, such as change requests.
Version control system	Application software that permits you to manage multiple revisions of the same file. It maintains the revision history that is generated as files evolve into their final forms.

View	A view, also called a project view, is a way of looking at a project. It enables you to see the parts of the project you need to see, without the confusion of seeing the entire project. A view consists of an application folder hierarchy and the items associated with each folder in that hierarchy. The name of a project and its root view are often identical. A view may or may not permit branching, be read only, or have a connection to its parent view.
View label	Serves as a time stamp for the entire contents of a view. Having a view label allows you to roll back a view to that label and see everything that existed at that point in time. Unless the view label is frozen, you can adjust it by attaching or detaching it from items. You can also move a view label from one revision to another.
Work around	Steps users must take to avoid a problem in the product. Workarounds are frequently offered until the resolution of a defect becomes available.
Working file	Any file in a working folder can be considered a working file. Often, these files have been checked out for modification. When checked in, a working file becomes a revision.
Working folder	Workstation folder to which copies of application files are checked out and from which files are added and checked in to the application folder.

# Index

## A

- access rights
  - child folder 145
  - folder 144
  - granting folder-level 143
  - granting item-level 147
  - item 148
  - promotion states 272
- active process items 255
- Annotate view 19
- annotations 232
- application preferences 118
- architecture 12
- artifacts
  - items 37
- atomic check-ins 246
- audit
  - fields 57
  - filters 276
- audit log
  - events 56
- Audit Log view 20

## B

- background operations
  - setting 34
  - viewing 34
- branching
  - effects on change requests 108
  - items 107
  - overview of options 106
  - understanding 106

## C

- change package
  - check-in 253
  - workspace 253
- change request
  - assign 180
  - branching effects 108
  - built-in workflow 43
  - close 182
  - comment 177
  - create 176
  - custom options 177
  - default fields 178
  - description 176
  - fields 178, 184
  - filters 275
  - link 182
  - mark read or unread 165
  - moving 180
  - moving effects 108
  - properties 183
  - required fields 178

- resolve 180
- sharing effects 108
- solution 177
- summary 176
- synopsis 176
- tracking system model 42
- verify 181
- viewing unread 178
- Change Request view 21, 174
- change requests
  - abbreviations 192
  - adding attachments 177
  - editing 178
  - lifecycle 182
  - reviewing linked 173
  - status field 183
- change status
  - converting 233
- Change view 21
- cheap copies 54
- check-in
  - atomic 246
  - exclusive 82
  - workspace change package 253
- check-in and out 7, 246, 250
- check-ins
  - editing comments 234
- check-out
  - statistics 251
- check-outs
  - folders 30, 249
  - keyword expansion 234
  - optimizing 250
- Classic perspective 16
- columns
  - display 33
- comments
  - editing 234
  - requirement 82
- common operations 60
- comparing
  - across views 160
  - by dates 160
  - file properties 160
  - folder properties 160
  - item properties 168
  - revisions 34, 159
  - to latest revision 159
  - to selected revision 160
  - with labels 159
- configuration
  - view 96
- configuring
  - folders 161
  - label properties 264
  - MPX profiles 32
  - product 31
- connections

- creating 31, 74
- logging on 75
- removing 32
- console preferences 121
- containers 35
- converting
  - change status 233
- copies 54
- copying
  - items 169
- creating
  - rich content fields 67
- cross-project
  - activity support 72
  - file dependencies 73

## D

- data
  - filtering 273
  - grouping 167
  - sorting 167
- date and time 64
- date comparison 160
- Detail view 23
- dialog box editors 28
- dot notation 51

## E

- editors
  - item 28
- embedded editors 28
- enterprise advantage license products 11
- enterprise license products 9
- EOL
  - character 252
  - conversion 78
- EOL conversion 235, 252, 253
- events
  - audit log 56
- external link 48, 169

## F

- field
  - content 63
  - date and time 64
  - enumerated 65
  - group 66
  - map 66
  - text 67
  - time span 68
  - user 69
- fields
  - audit 57
  - change request 184
  - display 33
  - file 237
  - folder 150
  - overview 29

- requirement 199
- task 210
- topic 223
- file
  - executable 232
  - fields 237
  - link and pin file revision to active process item 171
  - link and pin file revision to process item 171
  - linking 170
  - process item 170
  - process items 231
  - properties 236
  - revisions 233
- File Compare/Merge
  - comparing and merging files 233
  - overview 258
  - specifying as alternate utility 259
  - versions 258, 259
- filer
  - baseline 256
- files
  - associated 173
  - change status 233
  - checking out 248
  - comparing properties 160
  - comparing to latest revision 159
  - comparing to selected revision 160
  - hiding 163
  - locks 230
  - modifying 230
  - opening 230
  - overview 40, 228
  - project 81, 231
  - read-only 83, 232
  - renaming 230
  - replacing with label 166
  - replacing with view 166
  - rolling back to latest 166
  - rolling back to specific date 167
  - synchronizing 30, 76
  - version control 40, 228
- filter
  - apply 273
  - copying a filter 273
  - delete 274
  - edit 274
  - reset 275
- filters
  - audit 276
  - change request 275
  - overview 273
  - requirement 275
  - task 276
  - topic 276
- folder
  - access rights 144
  - change name or description 159
  - change status 233
  - configure 160
  - detach 268
  - fields 150
  - moving 165

- moving between two different views 166
- moving within same view 166
- open 166
- opening a local folder from a folder selection in StarTeam 166
- properties 148
- working 103, 157, 158

#### folders

- adding to views 157
- attach existing view or revision label to folder 266
- checking out 30, 249
- comparing properties 160
- configuring 161
- create revision label and attach to folder 265
- deleting 161
- designate to share 75
- hiding 163
- hierarchy 38, 141
- labels 265
- modifying configurations 161
- overview 38, 141
- replacing with label 166
- replacing with view 166
- rolling back 161
- rolling back to latest 166
- rolling back to specific date 167
- tree hierarchy 139
- views 39
- working with 157

## G

- glossary 283

## H

- History view 24

## I

- import 33
- internal link 48, 169
- item
  - access rights 148
  - attaching existing view or revision label to selected items 266
  - change name or description 159
  - configure 160
  - detach 269
  - label 266
  - moving 165
  - moving between two different views 166
  - moving within same view 166
  - properties 167, 172, 235
- item properties
  - comparing 168
- items
  - artifacts 37
  - comparing properties 168
  - editors 28
  - find 162

- flag 163
- linking 163, 169
- locking 234
- selecting 168
- working with 157

## K

- keyword expansion 78
- keywords 79

## L

### label

- attaching existing view or revision label to selected items 266
- copy 104, 267
- delete 268
- demote 267
- detach 268, 269
- freeze 269
- item 266
- promote 267
- revision 264, 266–268
- view 263

### label decorations

- preferences 128
- provider 129
- server explorer 130

- Label view 24

### labels

- attach existing view or revision label to a folder 266
- attaching to folders 158
- comparing 159
- configuring properties 264
- create revision label and attach to folder 265
- demoting 105
- folders 265
- managing 259

### license

- packages 8

### link

- delete 170
- enhanced 256
- external 48, 169
- file 231
- internal 48, 169
- link and pin file revision to active process item 171
- link and pin file revision to process item 171
- procedure 163, 169
- properties 172
- revisions 171
- to a specific revision 172

- Link views 25

- list of change requests 21, 174

- location references 162

### lock

- about 164
- exclusive 82
- using a menu 165
- using the toolbar 164

- logging off 15

## M

- mapping
  - preferences 126
- merging 51
- modal editors 28
- MPX
  - profile configuration 32

## N

- non-modal editors 28

## O

- options
  - view type 97

## P

- password management 131
- paths 140
- preferences
  - StarTeam 118
- personal options
  - change request 120
  - file 122
  - folder 125
  - MPX 133
  - requirements 131
  - sprint 132
  - story 134
  - task 135
  - topic 136
  - workspace 137
- personal preferences 118
- perspectives
  - Activity 15
  - Classic 16
  - opening 31, 100
- preferences
  - application 118
  - console 121
  - exporting team set 33
  - importing team set 33
  - label decorations 128–130
  - mapping 126
  - password management 131
  - personal 118
  - setting 31, 118
  - team set 118
- process items
  - active 81, 168, 255, 257
  - associated files 173
  - files 231
- process rules
  - baselines 256
  - project 80, 257
- product configuration 31
- product support 14
- project

- check out 30, 249
- create 76
- delete 77
- exclude files 81, 231
- name 77
- process rules 80, 257
- properties 78
- structure 70
- project properties
  - editors 87
  - name 85
  - options 85
  - overview 85
  - process rules 86
- projects
  - autonomy 71
  - creating 30, 76
  - designate to share 75
  - overview 69
  - refactoring 77
  - searching 83
  - sets 33
  - sharing workspace 29, 74
  - synchronizing 75, 84, 85
- promotion states
  - access rights 272
  - create 272
  - creating a new promotion state 272
  - deleting 273
  - editing 273
  - managing 259
- properties
  - change request 183
  - email 162
  - file 236
  - folder 148
  - modify 167, 235
  - provider 87, 155
  - requirement 198
  - task 217
  - topic 227
- property field
  - content 63
  - date and time 64
  - enumerated 65
  - group 66
  - map 66
  - text 67
  - time span 68
  - user 69
- property fields
  - creating rich content fields 67
  - displaying 63
- provider properties 87, 155

## Q

- queries
  - overview 273
- query
  - copy 277
  - create 278, 279

- delete 279
- edit 279
- options 281, 282
- predefined 280
- relational operators 280

quick diff 234

## R

- refactoring 77
- Reference view 25
- references
  - adding items to views 113
  - branching views 111
  - manually sharing objects 115
  - moving objects 116
  - overview 108
  - understanding 108
- replacing files or folders
  - with a label 166
  - with a view 166
- repositories
  - synchronizing 84
- requirement
  - create 196
  - field 199
  - filters 275
  - mark read or unread 165
  - properties 198
- Requirement view 25, 195
- revision
  - comments 82
  - linking 171
  - linking to a specific revision 172
- revision comparison 34, 159
- rollback 160
- rolling back
  - folders 161
  - to latest 166
  - to specific date 167

## S

- Search Results view 28
- server configurations
  - deleting 32
  - refreshing 34
- Share Project wizard
  - opening 74
  - using 29, 74
- sharing 54
- source control 12
- SupportLine 14
- synchronization 84

## T

- task
  - fields 210
  - filters 276
  - mark read or unread 165

- properties 217
- Task view 26
- tasks
  - about 47, 206
  - adding a work record 208
  - assigning resources 207
  - creating 206
  - deleting a work record 209
  - editing a work record 208
  - estimating 207
  - filtering 256
  - notes 208
  - work record 208
- templates
  - sample change request 192
  - sample file 235
  - sample requirement 197
  - sample task 209
  - sample topic 222
- terminology 283
- time span 68
- topic
  - creating 220
  - field 223
  - filters 276
  - mark read or unread 165
  - properties 227
  - responding to 221
- Topic view 27
- topics 47, 219

## U

- UI tour 15
- UNIX 232
- URLs
  - copying 85
- user-defined property fields
  - about 63

## V

- view
  - activity view 94
  - build 95
  - configuration 96
  - configuration and management 89
  - copy 104
  - delete 102
  - demote 267
  - label 263
  - labels 262
  - main view 94
  - overview 89
  - promote 267
  - properties 102
  - release view 95
  - roles 93
  - working folder 103, 158
- view configuration 96
- view labels
  - creating 103

- promoting 105
- view roles 93
- view type
  - options 97
- views
  - adding folders 157
  - Annotate 19
  - Audit Log 20
  - Change 21
  - Change Request 21, 174
  - comparing 160
  - designate to share 75
  - Detail 23
  - History 24
  - Label 24
  - Link 25
  - linking 32
  - opening 31, 100
  - opening existing 75
  - overview 17
  - proper use 96
  - Reference 25

- Requirement 25, 195
- Search Results 28
- Server Explorer 17
- switching between 104
- Task 26
- Topic 27
- types 91
- working with 100

## **W**

- wizard
  - Share Project 29, 74
- working folder
  - changing default 103, 158
  - creating 157
  - overview 142
- workspace
  - change package 253
  - projects 29, 74, 76