File Control System 1.0

Functional Specification

Author: Ken Rodham
Date: December 16, 2004
Revision: 1

Terminology

Repository

The directory/file structure containing the team's files is called the **repository**. It is assumed that the repository is accessible to all team members via a shared file system. The repository may contain subdirectories to any level of nesting, and each subdirectory may contain any number of files. The top-level repository directory is called the **repository root directory**.

Workspace

Each team member has a local (i.e., unshared) copy of the repository on his or her local computer. This local copy of the repository is called a **workspace**. A user's workspace is a mirror-image copy of the real repository, containing the same directories and files. A user initially creates their workspace by asking FCS to copy the repository to a specified location on their local computer. This location is called the **workspace root directory**. The user can make changes to files locally in their local workspace, and then, when the changes are complete, ask FCS to copy the changes back to the repository and make them available to the rest of the team.

File Revisions

FCS keeps track of all **revisions** made to repository files. For each file, revisions are numbered sequentially starting at 1. The initial copy of a file that is added to the repository becomes revision 1, the next update to the file becomes revision 2, and so on. While FCS keeps track of file revisions, it does not keep track of directory revisions.

Configuration

In order to use FCS, a user must provide the following configuration information:

1. The full path of the repository root directory. This should be the same for all team members
2. The full path of their workspace root directory. This may vary between team members because each member may put their workspace at a different location in their local file system. Each member must manually create their workspace root directory using regular file system commands

Given this information, FCS can copy files and directories between the user's workspace and the repository.

Use Case: Configure FCS

1. The user specifies the following information:
    a. The full path of the workspace root directory. Example:
    `/home/fred/cs248/workspace/`
    b. The full path of the repository root directory. Example:
    `/shared/cs428/repository/`

NOTE: All remaining use cases assume that the user has previously performed the Configure FCS use case.

Repository Initialization

Before the repository can be used, the FCS administrator must:

1. Create the repository root directory using regular file system commands
2. Run FCS and tell it to initialize the repository

Once repository initialization has been complete, the repository is ready to be accessed by team members. Initialization is an administrative function that is performed only once before a repository is used for the first time.

Use Case: Initialize Repository

This use case allows a user to initialize the repository.

1. The user requests repository initialization
2. All initialization operations that are needed to prepare the repository for use are performed
3. A successful completion message is displayed

Variations:

1. If the repository directory does not exist, an error message is displayed and the operation is aborted
2. If the repository directory is not empty, an error message is displayed and the operation is aborted
3. If initialization fails, an error message is displayed and the operation is aborted

Adding Files/Directories to the Repository

After repository initialization, the repository contains one directory (the root) and no files. Additional files/directories to be added to the repository are first created in the user's local workspace. When the user is ready to share the new files/directories, they perform an FCS add operation to add them to the repository. Adding a file/directory to the repository creates a copy of it in the repository, but does not remove it from the user's workspace.

Files/directories are added to the repository one at a time. There is no batch mode for adding multiple files/directories at once. When adding a file, the user must provide a textual description of the file's contents.

FCS can handle text files and binary files. When adding a file to the repository, the user must specify whether the file is a text file or a binary file.

Use Case: Add File

This use case allows users to add workspace files to the repository.

1. The user specifies the name of a workspace file, the type of the file (text or binary), and a string describing its contents, and requests that the file be added to the repository
2. The workspace file is added to the repository as revision 1
3. If the file is a text file, keyword substitution is performed on both the workspace and repository copies of the file (see the Keyword Substitution section)
4. A successful completion message is displayed

Variations:

1. If the workspace file does not exist, an error message is displayed and the operation is aborted
2. If the file already exists in the repository, an error message is displayed and the operation is aborted
3. If the file's directory does not exist in the repository, an error message is displayed and the operation is aborted
4. If adding the file fails for any reason, an error message is displayed and the operation is aborted

Use Case: Add Directory

This use case allows users to add workspace directories to the repository.

1. The user specifies the name of a workspace directory and requests that it be added to the repository
2. The workspace directory is added to the repository
3. A successful completion message is displayed

Variations:

1. If the workspace directory does not exist, an error message is displayed and the operation is aborted
2. If the directory already exists in the repository, an error message is displayed and the operation is aborted
3. If the directory's parent directory does not exist in the repository, an error message is displayed and the operation is aborted
4. If adding the directory fails for any reason, an error message is displayed and the operation is aborted

Checking Out Files from the Repository

When a user wants to retrieve the latest revision of a file from the repository, they must perform an FCS checkout operation. A checkout operation copies the latest revision of the specified file from the repository into the user's workspace. If the user wants to checkout an earlier revision of the file rather than the latest one, they may specify the desired revision number when checking out the file.

If a user wants to checkout a repository file with the intent of changing it, they must checkout the file in locked mode. FCS only allows one user to lock a particular repository file at a time, thus preventing multiple users from simultaneously modifying the file. When checking out a file in locked mode, it is possible that the file has already been locked by another user, in which case FCS will notify the user that the file is already locked. Only the latest (or current) revision of a file may be checked out in locked mode. Earlier revisions may not be modified.

Use Case: Checkout File

This use case allows a user to copy a specified revision of a repository file into their workspace.

1. The user specifies the name of the repository file and optionally a revision number or label, and requests that the file be checked out. If the file is being checked out in locked mode, this is also specified. If no revision number or label is specified, it defaults to the current revision. If a label is specified, the revision number corresponding to that label is checked out (see the Repository Labeling section)
2. The specified revision of the repository file is copied into the user's workspace. If the file is being locked, the repository file is also marked as being locked so that no other users can lock it
3. A successful completion message is displayed

Variations:

1. If the repository file does not exist, an error message is displayed and the operation is aborted
2. If the specified revision of the repository file does not exist, an error message is displayed and the operation is aborted
3. If no revision of the repository file has the specified label, an error message is displayed and the operation is aborted
4. If locked mode is specified and the repository file is already locked by another user, an error message is displayed and the operation is aborted. The error message indicates which user has the file locked
5. If locked mode is specified and a revision other than the current revision is requested, an error message is displayed and the operation is aborted (only the current revision can be checked out in locked mode)
6. If the checkout process fails for any reason, an error message is displayed and the operation is aborted

Checking Out Directories from the Repository

In addition to checking out files, a user may also checkout an entire repository directory. Checking out a repository directory recursively copies all of the files/directories in the specified repository directory to the user's workspace. This makes it convenient for users to retrieve the latest revisions of all files/directories in the repository or some part thereof. An example of this is a new team member who needs to copy all of the repository files/directories to their newly created workspace. After creating their workspace root directory on their local file system, the user would perform an FCS checkout operation on the repository root directory, thereby copying all files/directories from the repository to the workspace.

Use Case: Checkout Directory

This use case allows a user to recursively checkout all of the files and subdirectories in a specified repository directory. By default, every subdirectory and the current revision of every file within the specified repository directory are copied to the workspace. Alternatively, the user may specify a label indicating which version of the repository directory should be checked out. If a label is specified, only those files and subdirectories that have the label are copied to the workspace. It is an error to specify a label that is not associated with the specified repository directory. However, it is expected that some files and subdirectories within the repository directory will not have the specified label. This is not an error, but files and subdirectories that do not have the label are not copied to the workspace. This allows the user to checkout only those files and directories that are part of the specified version of the repository.

A directory may not be checked out in locked mode.

1. The user specifies the name of the repository directory and optionally a label, and requests that the directory be checked out
2. If the corresponding workspace directory does not exist, it is created
3. For each file in the specified repository directory
    a. If a label was specified
        i. If the file has the specified label, copy the labeled revision of the file to the workspace
        ii. If the file does not have the specified label, do not copy the file to the workspace at all
    b. If a label was not specified, copy the current revision of the file to the workspace
4. For each subdirectory in the specified repository directory
    a. If a label was specified
        i. If the subdirectory has the specified label, recursively copy the subdirectory to the workspace
        ii. If the subdirectory does not have the specified label, do not copy the subdirectory to the workspace
    b. If a label was not specified, recursively copy the subdirectory to the workspace

5. A successful completion message is displayed

Variations:

1. If the repository directory does not exist, an error message is displayed and the operation is aborted
2. If the specified repository directory does not have the specified label, an error message is displayed and the operation is aborted
3. If directory creation fails, and error message is displayed and the operation is aborted.

Checking In Modified Files

After checking out a file in locked mode, the user is free to make changes to the file. When the changes are complete, the new revision of the file may be copied to the repository by executing an FCS checkin operation. If the latest revision of the file is N, the new revision becomes revision N+1. When checking in a file, the user must provide a textual log message describing the changes that were made to the file in the new revision. Checking in a file copies the new revision to the repository, but does not remove it from the user's workspace.

Use Case: Checkin File

This use case allows a user to checkin a file that they currently have locked.

1. The user specifies the name of a workspace file and a log message describing the changes that have been made to the file since it was checked out, and requests that the file be checked in
2. The workspace file is copied into the repository as revision N+1, and the user's lock on the repository file is released
3. If the file is a text file, keyword substitution is performed on both the workspace and repository copies of the file (see the Keyword Substitution section)
4. A successful completion message is displayed

Variations:

1. If the specified workspace file does not exist, an error message is displayed and the operation is aborted
2. If the corresponding repository file does not exist, an error message is displayed and the operation is aborted
3. If the user does not hold the lock on the repository file, an error message is displayed and the operation is aborted
4. If the checkin process fails for any reason, an error message is displayed and the operation aborted

Unlocking Files

If a user locks a file for editing and then decides not to change the file after all, they should execute an FCS unlock operation to release their lock on the file. Doing so will allow other users to modify the file. Unlocking a file does not remove it from the user's workspace.

Use Case: Unlock File

This use case allows a user to unlock a file that they currently have locked.

1. The user specifies the name of a repository file and requests that it be unlocked
2. The user's lock on the specified repository file is released
3. A successful completion message is displayed

Variations:

1. If the specified repository file does not exist, an error message is displayed and the operation is aborted
2. If the user does not hold the lock on the specified repository file, an error message is displayed and the operation is aborted
3. If releasing the lock fails for any reason, and error message is displayed and the operation is aborted

Removing Files/Directories from the Repository

Repository files and empty repository directories that are no longer needed may be removed from the repository by performing an FCS remove operation. Files/directories are removed from the repository one at a time. There is no batch mode for removing multiple files/directories at once. Removing a file/directory from the repository does not remove the corresponding file/directory from the user's workspace. If the user wants to remove the workspace copy, they may do so using regular file system commands.

Use Case: Remove File

This use case allows users to remove files from the repository.

1. The user specifies the name of a repository file and requests that it be removed from the repository
2. The repository file is removed from the repository
3. A successful completion message is displayed

Variations:

1. If the repository file does not exist, an error message is displayed and the operation is aborted
2. If the repository file is locked, an error message is displayed and the operation is aborted. The error message indicates which user has the file locked
3. If the repository file has been labeled, an error message is displayed and the operation is aborted (see the Repository Labeling section)
4. If removing the file fails for any reason, an error message is displayed and the operation aborted

Use Case: Remove Directory

This use case allows users to remove directories from the repository.

1. The user specifies the name of a repository directory and requests that it be removed from the repository
2. The repository directory is removed from the repository
3. A successful completion message is displayed

Variations:

1. If the repository directory does not exist, an error message is displayed and the operation is aborted
2. If the repository directory is not empty, an error message is displayed and the operation is aborted
3. If the repository directory has been labeled, an error message is displayed and the operation is aborted (see the Repository Labeling section)
4. If removing the directory fails for any reason, an error message is displayed and the operation is aborted

Repository Labeling

FCS supports repository labeling for keeping track of which files and directories make up a particular version of the repository. When the repository is labeled, the user specifies a label string that describes the current version of the repository (e.g., “BETA_1”, “VERSION_1”, “VERSION_1.1”, etc.). Every directory and the most recent revision of every file in the repository is marked with the specified label. Thereafter, if there is ever a need to reconstitute that particular version of the repository, it can be done by specifying the correct label during an FCS checkout operation (e.g., “checkout VERSION_1.1”). Doing so would cause all of the files and directories that make up the specified repository version to be copied into the user’s workspace.

Every repository directory and file revision is marked with zero or more labels. If a directory or file revision has a particular label, then it is part of that version of the repository.

Any file or directory that has been marked with one or more labels may not be removed from the repository using an FCS remove operation because doing so would make it impossible to reconstitute previous versions of the repository.

Any non-empty string may be used as a label.

Use Case: Label Repository

This use case allows a user to label all of the files and directories in the repository with a descriptive label.

1. The user specifies a label string and requests that the repository be labeled
2. Every directory and the current revision of every file in the repository is marked with the specified label string

Variations:

1. If the user specifies an empty label string, or a label string that has already been used, an error message is displayed and the operation is aborted
2. If the repository cannot be labeled for any reason, an error message is displayed and the operation is aborted

Viewing Repository, Directory, and File Information

Users may ask FCS to display information about the repository itself or about a particular file/directory within the repository.

Repository information includes:

- The name of the user who initialized the repository
- The date/time at which repository initialization occurred
- A list of all labels that have been applied to the repository. For each label, the following information is displayed:
  1. label string
  2. name of the user who created the label
  3. date/time at which the label was created

Directory information includes:

- The full path of the directory within the repository
- The name of the user who added the directory to the repository
- The date/time at which the directory was added to the repository
- A list of all labels that have been applied to the directory

File information includes:

- The full path of the file within the repository
- A description of the file's contents (provided by the user when the file was added to the repository)
- The name of the user who added the file to the repository
- The date/time at which the file was added to the repository
- The current revision number of the file
- The name of the user who currently has the file locked (if any)
- The date/time at which the user locked the file (if any)
- A list of all labels that have been applied to the file. For each label, the following information is displayed:
  1. label string
  2. file revision number that is associated with the label

Use Case: View File Information

This use case allows users to view information about a repository file.

1. The user specifies the name of a repository file and requests that information about the file be displayed
2. The following information about the repository file is displayed:
   a. full repository path
   b. description string (provided when the file was added to the repository)
   c. creation user
   d. creation date/time
   e. current revision number

f. lock user (only if the file is currently locked)
g. lock date/time (only if the file is currently locked)
h. A list of all labels that have been applied to the file. For each label, the following information is displayed:
    i. label string
    ii. file revision number that is associated with the label

Variations:

1. If the specified repository file does not exist, an error message is displayed and the operation aborted
2. If the file information cannot be displayed for any reason, an error message is displayed and the operation is aborted

Use Case: View Directory Information

This use case allows users to view information about a repository directory.

1. The user specifies the name of a repository directory and requests that information about the directory be displayed
2. The following information about the repository directory is displayed:
    a. full repository path
    b. creation user
    c. creation date/time
    d. A list of all labels that have been applied to the directory

Variations:

1. If the specified repository directory does not exist, an error message is displayed and the operation aborted
2. If the directory information cannot be displayed for any reason, an error message is displayed and the operation is aborted

Use Case: View Repository Information

This use case allows users to view information about the repository.

1. The user requests that information about the repository be displayed
2. The following information about the repository is displayed:
    a. creation user
    b. creation date/time
    c. A list of all labels that have been applied to the repository. For each label, the following information is displayed:
        i. label string
        ii. name of the user who created the label
        iii. date/time at which the label was created

Variations:

1. If the repository information cannot be displayed for any reason, an error message is displayed and the operation is aborted

Viewing a File's Revision History

Users may ask FCS to display the revision history of a repository file. For each revision of the file, FCS will display the following information:

- The revision number
- The name of the user who made the revision
- The date/time at which the revision was made
- The log message provided by the user when the revision was checked in

Use Case: View File Revision History

This use case allows the user to view the revision history of a repository file.

1. The user specifies the name of a repository file and requests that the file's revision history be displayed
2. The revision history of the file is displayed in reverse chronological order. For each revision of the file, the following information is displayed:
    a. revision number
    b. revision user
    c. revision date/time
    d. revision log message (provided when the file was checked in)

Variations:

1. If the specified repository file does not exist, an error message is displayed and the operation aborted
2. If the file revision history cannot be displayed for any reason, an error message is displayed and the operation is aborted

Viewing File Revision Differences

Users may ask FCS to display the differences between two revisions of a text file. FCS compares the two file revisions and, at each point where the revisions differ, displays the lines that were inserted, deleted, or modified between the two revisions (similar to the Unix `diff` command).

Use Case: View File Revision Differences

This use case allows the user to view the differences between two revisions of a text file.

1. The user specifies the name of a repository file, along with the numbers of the two revisions of the file that are to be compared
    a. if no revision numbers are specified, FCS compares the copy of the file in the user's workspace with the current revision of the file
    b. if the user specifies one revision number, FCS compares the copy of the file in the user's workspace with the specified revision of the file
    c. if the user specifies two revision numbers, FCS compares the two specified revisions of the file
2. The differences between the two specified revisions are displayed (i.e., lines inserted, deleted, or modified between the two revisions)

Variations:

1. If the specified repository file does not exist, an error message is displayed and the operation aborted
2. If the specified repository file is a binary file, an error message is displayed and the operation aborted
3. If one or both of the specified file revisions does not exist, an error message is displayed and the operation aborted

Keyword Substitution

FCS performs automatic keyword substitution on text files whenever they are added or checked-in to the repository. This allows revision control-related information to be embedded within the body of a text file. The following keywords are supported:

@Path@ - the path of the file relative to the repository root
@Revision@ - the revision number of the file
@Author@ - the name of the user who checked in this revision of the file
@Time@ - the date/time at which this revision of the file was checked in

For example, the initial creator of a text file could include the following header in the file:

```
/*
@Path@
@Revision@
@Author@
@Time@
*/
```

When the file is added to the repository, FCS would substitute each of the keywords as follows:

```
/*
@Path: project/src/SomeFile.java @
@Revision: 1 @
@Author: fred @
@Time: Fri Jun 13 12:00:29 MDT 2003 @
*/
```

Keyword substitution is performed on both the workspace and repository copies of the file. When revision 2 of the file is checked in, FCS would make the following substitution in both the workspace and repository:

```
/*
@Path: project/src/SomeFile.java @
@Revision: 2 @
@Author: barney @
@Time: Mon Jun 16 10:34:13 MDT 2003 @
*/
```

Keyword substitution is only performed on text files, not on binary files. Keywords need not appear in comments; they may appear anywhere in a text file. In fact, any text file may contain keywords, not just source code files.