Overview

As more software development teams move to IBM® Rational Team Concert™ web-based collaboration software for managing the full software development lifecycle, concerns about migrating historical data can be a consideration. Many projects have been running for years that use various other software configuration management (SCM) systems. Therefore, there is a huge amount of historical data in those SCM systems. How to migrate that data to Rational Team Concert is a top challenge for those project teams.

This tutorial is based on our experience in migrating historical data from Configuration Management Version Control (CMVC) to the Rational Team Concert online system. It describes how to export the history data from CMVC and import it into Rational Team Concert without losing the links between various code artifacts. The migration steps outlined, using the CMVC client APIs and the Rational Team Concert APIs, are also helpful for the migration from any other SCM systems to Rational Team Concert projects.

Comparison of CMVC and Rational Team Concert

CMVC basics

Configuration Management Version Control (CMVC) is a well-known software configuration management (SCM) product. It integrates four facets of the software development process in a distributed development environment to facilitate project-wide coordination of development activities across all phases of the product development life cycle:
Using the Jazz API to migrate historical data from CMVC to Rational Team Concert projects

- Configuration management
- Version control
- Change control
- Problem tracking

In CMVC, the following concepts and objects are introduced for the source code management:

- Components provide organization and control of development data.
- Releases contain files for product-related activities.
- Defects are used to record information about reported problems.
- Features are used to record information about proposed enhancements.

**Rational Team Concert basics**

Rational Team Concert is one of the next-generation collaborative application life cycle management products. It operates on the IBM® Rational® Jazz™ technology platform. Jazz is a scalable, extensible team-collaboration platform that integrates tasks across the software life cycle.

Rational Team Concert makes it easy to exchange information and collaborate with your team in the context of your project. If a change request arrives, you and other team members are notified of the change automatically. You can refer to the change in chat sessions and link it to the related artifacts. Business stakeholders can also automatically stay informed about the status of task changes that interest them.

Many aspects of the software development life cycle are integrated seamlessly in a single environment, including agile development planning, process definition, source control, defect report tracking, build management, and reporting. You can track and manage the links and associations between different artifacts, promote sound development processes, and gather project information automatically. Rational Team Concert enables process configuration and customization and provides users with an Eclipse-based client interface, a Microsoft® Visual Studio client interface, and a web interface.

**Comparison of the two**

Both CMVC and Rational Team Concert provide powerful SCM functions, but Rational Team Concert puts more focus on team cooperation and collaboration. Therefore, it makes development and project management more efficient.

Compared to CMVC, Rational Team Concert introduces similar terms for the source control, such as component, defect, and feature (or story). However, it's important to be aware that stream is used in Rational Team Concert similar to CMVC, but with several additional capabilities. In addition, the Rational Team Concert workspace serves as an area where you can view and modify components.

**Migration process overview**

Although the configuration management, version control, change control, problem tracking, and data storage are rather complex in both CMVC and Rational Team Concert, luckily, we don't need to worry about them. We just need to have an idea about the difference between the CMVC and
Rational Team Concert data formats and to become familiar with the CMVC and Rational Team Concert clients, and then automate the repeating steps of export and import by calling CMVC client APIs and Jazz APIs.

In this approach, you have full control of the migration process. If any error occurs, it is easy to fix the problem and restart the migration from where the error occurred. For example, if the remote Rational Team Concert server is unavailable for some reason while you are importing source files, it is safe to find the last successfully imported file and then re-import from the next source code file.

There are typically three steps, which are illustrated in Figure 1:

1. Preparation
2. Export and import
3. Verification
Prerequisite

Migration environment

The CMVC server version is 5.0 and the Rational Team Concert server version is 2.0 (Scrum Template 2.0).

CMVC and Rational Team Concert libraries
• For CMVC 5.0, you can find the CMVC.jar file in the CMVC client installation folder.
• For Rational Team Concert Version 2.0, you need follow this guide to set up the development environment: Integrating and Extending Rational Team Concert 2.0 (the SDK)

Migration user access

• For CMVC 5.0, the user needs the privilege to view defects and features and to extract source code files.
• For Rational Team Concert v2.0 or later, the user should have the Scrum Master role in project area.

Focus of this tutorial

The objects between CMVC and Rational Team Concert don't match 100% with each other, so we can pay attention to only the data that we are working with. For instance, we could decide just to migrate some fields of defects or features that matter to your project, rather than all the fields.

For this tutorial, we focus on performing the following repetitive steps in an automatic manner by calling CMVC client APIs and Rational Team Concert client APIs:

• Export defect or feature (defect/feature)
• Export source code and history version information
• Import defect/feature
• Import source code

Prepare for the migration process

Prepare the Rational Team Concert repository

We need to set up the target stream, workspace, components, work item categories, as well as custom attributes in Rational Team Concert prior to the migration (most of the preparation steps that follow can also be done by calling Rational Team Concert APIs, but that is not the focus of this article). To do so, we will complete these tasks:

1. Create a stream named releaseA as the target stream.
2. Add two components to releaseA: src and build. The components are based on the CMVC component names.
3. Create a workspace, releaseAWorkSpace, and target releaseAWorkSpace to the releaseA stream.
4. Add the work item categories: src and build. These are based on the CMVC component names.
5. Add the custom attribute, cmvc_name, for storing the CMVC defect and feature number. Usually, it is a good practice to keep the original CMVC defect and feature number. We can do that by introducing a custom attribute to Rational Team Concert. Add a custom attribute for the defect and task work item types to Rational Team Concert. The custom attribute name is of the small string type.
Object mapping rules

This section will deal with the CMVC and Rational Team Concert object mapping rules. These mapping rules work for most of the cases, but you can adapt them to your projects as necessary. We need to take into account the mappings for users, defects and features, attributes, and terms, so the following seven tables cover each of those individually.

Table 1. User mapping

<table>
<thead>
<tr>
<th>Rational Team Concert user</th>
<th>CMVC user</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmailAddress</td>
<td>Address</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>UserId</td>
<td>user_id</td>
</tr>
</tbody>
</table>

Note: User_id is used as the identifier in CMVC and will be mapped to UserId in Rational Team Concert.

Table 2. Defects and features mapping

<table>
<thead>
<tr>
<th>Attribute type</th>
<th>CMVC defect or feature</th>
<th>Rational Team Concert work item type: defect or ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default attributes</td>
<td>Name</td>
<td>cmvc_name (custom attribute on Rational Team Concert side )</td>
</tr>
<tr>
<td></td>
<td>compName</td>
<td>Filed Against</td>
</tr>
<tr>
<td></td>
<td>Abstract</td>
<td>Summary</td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td>Comments</td>
</tr>
<tr>
<td></td>
<td>remarks_seq</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td>State</td>
<td>Status/resolution</td>
</tr>
<tr>
<td></td>
<td>addDate</td>
<td>Creation Date</td>
</tr>
<tr>
<td></td>
<td>ownerName</td>
<td>Owned by</td>
</tr>
<tr>
<td></td>
<td>originName</td>
<td>Created by</td>
</tr>
<tr>
<td>Attribute specific to Defect</td>
<td>Priority</td>
<td>Priority</td>
</tr>
<tr>
<td></td>
<td>Severity</td>
<td>Severity</td>
</tr>
</tbody>
</table>

Note: This cmvc_name custom attribute will be used to store the CMVC defect or feature number. We can also map other attributes that we are interested in.

Table 3. Severity mapping

<table>
<thead>
<tr>
<th>CMVC</th>
<th>Rational Team Concert</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: critical</td>
<td>Critical</td>
</tr>
<tr>
<td>2: severe</td>
<td>Major</td>
</tr>
<tr>
<td>3: wrong</td>
<td>Normal</td>
</tr>
<tr>
<td>4: minimal</td>
<td>Minor</td>
</tr>
<tr>
<td>Any other value</td>
<td>Unclassified</td>
</tr>
</tbody>
</table>
### Table 4. Priority mapping

<table>
<thead>
<tr>
<th>CMVC</th>
<th>Rational Team Concert</th>
</tr>
</thead>
<tbody>
<tr>
<td>mustfix</td>
<td>HIGH</td>
</tr>
<tr>
<td>candidate</td>
<td>LOW</td>
</tr>
<tr>
<td>Deferred</td>
<td>LOW</td>
</tr>
<tr>
<td>easy</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>moderate</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>Difficult</td>
<td>HIGH</td>
</tr>
<tr>
<td>n/a</td>
<td>Unassigned</td>
</tr>
</tbody>
</table>

### Table 5. Defect states mapping

<table>
<thead>
<tr>
<th>CMVC Defect</th>
<th>Rational Team Concert Work Item (Defect)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>New</td>
</tr>
<tr>
<td>Working</td>
<td>In-Progress</td>
</tr>
<tr>
<td>Design</td>
<td>In-Progress</td>
</tr>
<tr>
<td>Size</td>
<td>In-Progress</td>
</tr>
<tr>
<td>Review</td>
<td>In-Progress</td>
</tr>
<tr>
<td>Returned</td>
<td>Resolved - Works for Me</td>
</tr>
<tr>
<td>Verify</td>
<td>Resolved - Fixed</td>
</tr>
<tr>
<td>Canceled</td>
<td>Verified - Works for Me</td>
</tr>
<tr>
<td>Closed</td>
<td>Verified - Fixed</td>
</tr>
<tr>
<td>Transferred</td>
<td>Verified - Fixed</td>
</tr>
</tbody>
</table>

### Table 6. Feature states mapping

<table>
<thead>
<tr>
<th>CMVC Feature</th>
<th>Rational Team Concert Work Item (Task)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>New</td>
</tr>
<tr>
<td>Working</td>
<td>In-Progress</td>
</tr>
<tr>
<td>Returned</td>
<td>Deferred</td>
</tr>
<tr>
<td>Verify</td>
<td>Implemented</td>
</tr>
<tr>
<td>Canceled</td>
<td>Invalid</td>
</tr>
<tr>
<td>Closed</td>
<td>Done</td>
</tr>
<tr>
<td>Transferred</td>
<td>Done</td>
</tr>
<tr>
<td>Design</td>
<td>In-Progress</td>
</tr>
<tr>
<td>Size</td>
<td>In-Progress</td>
</tr>
<tr>
<td>Review</td>
<td>In-Progress</td>
</tr>
</tbody>
</table>

### Table 7. Other concept mappings

<table>
<thead>
<tr>
<th>CMVC</th>
<th>Rational Team Concert</th>
</tr>
</thead>
</table>

Using the Jazz API to migrate historical data from CMVC to Rational Team Concert projects
Release | Stream
---|---
Component | Component
Changes | Change set

**Note:** The release in CMVC can be mapped to the stream in the Rational Team Concert project.

**Tips:**

- Please make sure that the version of the Rational Team Concert SDK and the version of the Rational Team Concert server are the same. Otherwise, unexpected exceptions might occur.
- The source code is included with the Rational Team Concert SDK. Looking into the code is a good way to pinpoint some problems. Finding help and information from the Jazz.net forum is also very helpful. You can find the information on the Jazz API the Jazz development wiki.
- The parameters of CMVC client APIs might have some tiny differences, depending on your exact CMVC version. Of course, it is easy to adjust them in the exporting code in the Downloadable resources section.

Export historical data from CMVC by using the API

**CMVC Java client API overview**

The CMVC Java™ client API provides a Java interface for client applications to access CMVC server functionality. All CMVC graphical user interface clients are now based on this API. Developers can now access CMVC by the API. The CMVC client API design provides units of interaction known as *commands*. Each of these is a logical equivalent of an already-existing CMVC command (such as the `user-view` command).

In this section, we will look briefly at the CMVC command innovation and then export the users, the defect and features, and the source files with history versions.

1. Invoke the CMVC API Java Archive (JAR) file to view file information:
   a. Run the following commands, as shown in Figure 2, and log in the CMVC:

   ```
   C:\IBM\CMVCDC50>set_classpath=CMVC.jar
   C:\IBM\CMVCDC50>java_com.ibm.sdwb.cmvc.client.api.SampleExe
   ```

   b. Then call the `FileView` command shown in Figure 3. We can get file information like what we can get in CMVC GUI client.
Using the Jazz API to migrate historical data from CMVC to Rational Team Concert projects

Figure 2. Log in to CMVC by using the CMVC API JAR file

```
C:\IBM\CMVCDC50>set classpath=CMVC.jar
C:\IBM\CMVCDC50>java com.ibm.sdwcmvcmvc.client.apil.SampleExec
********** CMVC Java Client API Demo **********
********** Enter Session Data **********
Enter family host name >> ausaix25.swo.xxxxxxx.ibm.com
Enter family port number >> 7677
Enter user name >> zhenghl
Enter authentication method (HOST/PW) >> PW
Enter password >> ********
Enter local source/destination directory >> E:\cmvc_root

********** Setup Information **********
Family host: ausaix25.swo.xxxxxxx.ibm.com
Family port: 7677
User name: zhenghl
Auth method: PW
Password: ********
Local dir: E:\cmvc_root

Proceed with these values (P) or re-enter (R)? >> P

********** Family Connected Successfully **********
Attempting to contact the family...
********** Family Connected Successfully **********
Host: ausaix25.swo.xxxxxxx.ibm.com
Port: 7677
Server Version: 54349

Attempting to log in...
login SUCCBED!!!
```
2. Call the CMVC client APIs in the Java program to view the file information. Calling the CMVC client APIs in the Java program is as simple as invoking the command.

You need to familiarize yourself with only a few Java classes provided by the CMVC client APIs. The Command Subsystem includes these classes:

**CommandFactory**
This is a mechanism for easy creation of Command objects.

**Command**
This is an abstract class encapsulating data related to a logical, or high-level, CMVC command.

**CommandResults**
Instances of this class encapsulate results generated by executing a command.

**SessionData**
Instances of this class carry session-specific data, such as that needed for authentication to CMVC families, as well as any parameter defaults.
FamilyInfo

Instances of this class encapsulate information needed to contact a given CMVC family.
The core java code fragment for calling CMVC FileView API is shown in Listing 1. You can get the complete code in the Download section.

Listing 1. The core Java code fragment for calling the FileView command

```java
// File -view
// get command
Command cmd = CommandFactory.getInstance().getCommand("FileView");
cmd.setFamilyInfo(familyInfo_);
cmd.setSessionData(sessionData_);

// set the command parameters
addParametersFileView(cmd, fileInfo.getFilepath());

// execute the command
CommandResults res = cmd.exec();

// analyze the essential information
int rc = res.getReturnCode();
if (rc == CommandResults.SUCCESS) {
    logger.info("FileView:Succes");
    String pathname = res.getValue("FileView", "pathName");
    String globaltext = res.getValue("FileView", "null");

    // analyze the key information: source code version, version owner and
    // the mapping defect/feature
    getFileVersionInfo(fileInfo, pathname, globaltext, release);
    // record the source code version, version owner and
    // the mapping defect/feature
    fileWriter.write(fileInfo.getFilepath() + "\t" + fileInfo.getHistoryInfo() + "\r\n");
}
else {
    logger.info("FileView:failed");
    logger.error(res.getFailureMessages());
}
```

As you now have an idea about the CMVC API use, we will move on to export the data from CMVC.

Export CMVC user data

In this case, we migrate only three fields for user information: User ID, User Name, and Email. It is very easy to export such user information directly from CMVC GUI client or commands.

1. We define a simple user data format:
   `userid \t user name \t email`
2. Change the exported user information to the format shown in Listing 2, and save to a text file called `User.txt`. For example:
   Jim Jhoon, Zon jjz@01.ibm.com jjz

Listing 2. Exported user information

```
Jim Jhoon, Zon jjz@01.ibm.com jjz
Hou liang, Zhen zhl@01.ibm.com zhl
Li Eric, Hans heh@01.ibm.com heh
Velu Lubin, Green vlg@01.ibm.com vlg
Kate Morton, David kmd@01.ibm.com kmd
```
Export the CMVC defect or feature

We need to get the entire lists for defect and feature numbers, respectively, before exporting the detailed defects and features. That can be done easily with the CMVC GUI client or commands.

1. Save the defect numbers to a text file named `Def_no_list.txt`, as shown in Listing 3.

Listing 3. Exported defect number list

| 19484 |
| 19838 |
| 19840 |
| 19866 |
| 19985 |
| 19986 |
| 19921 |
| 19943 |

2. Query the feature list in the CMVC GUI client, and save the entire feature numbers to a text file, `Fea_no_list.txt`, as shown in Listing 4.

Listing 4. Exported feature number list

| 20226 |
| 19428 |
| 19167 |
| 19405 |
| 21318 |
| 20286 |

3. Export defect or feature details in Java program

4. For each defect name in the Def_no_list.txt file, call the `DefectView` command in the Java program to get the defect detail information. Save it in your own format.

5. For each feature name in the Fea_no_list.txt file, call the `FeatureView` command in Java program to get the feature detail information and save it in your own format. You can find the complete code in the Download section.

Figure 4 illustrates the detailed content of a defect that we store in a text file.
Using the Jazz API to migrate historical data from CMVC to Rational Team Concert projects

Figure 4. The defect or feature storage format used for this article

Export the CMVC source code

1. We need to generate the source file path list for all the files stored in CMVC by using the CMVC GUI client or commands, and then save all of those file paths to a text file, File_path_list.txt, as Listing 5 shows.

Listing 5. Exported file path list

```
build/build.xml
build/make.cmd
src/lib/db2jcc.jar
src/lib/j2ee.jar
src/rad_echat/src/com/ibm/lenovo/database/driver/LenovoDBManager.java
src/rad_echat/src/com/ibm/lenovo/database/driver/TestDB.java
src/rad_echat/src/com/ibm/lenovo/database/driver/TestDBCreate.java
src/rad_echat/WebContent/chat/ChatView_01252007.jsp
src/rad_echat/WebContent/chat/ChatView_12112006.jsp
src/rad_echat/WebContent/chat/ChatView.jsp
```

2. Export source code in the Java program:
   a. For each source code file path in the File_path_list.txt file, call the FileView command in the Java program to get the history version information.
   b. For each version of a file, call the FileExtract command in the Java program.
   c. Save the specific history version source code content and the history version information in your own format. See the Downloadable resources section for the complete code.
Listing 6 shows the history version information storage format. The version information is saved in the Path_historyver.txt file, which will be used when importing the history versions to Rational Team Concert.

**Listing 6. Exported file history version information**

```plaintext
build/build.xml 1.1,jyoungmn,19428;1.2,jyoungmn,19428;1.3,elubin,19405;1.4,
    jyoungmn,19485;1.5,dmorton,20286;1.6,dmorton,21318
build/make.cmd 1.1,jyoungmn,19167;1.2,jyoungmn,19428;1.3,jyoungmn,19405
src/lib/db2jcc.jar 1.1,eggerh,28279
src/lib/j2ee.jar 1.1,jyoungmn,19428
src/rad_echat/src/com/ibm/lenovo/database/driver/LenovoDBManager.java 1.1,ZSHAE,20226
src/rad_echat/src/com/ibm/lenovo/database/driver/TestDB.java 1.1,ZSHAE,20226
src/rad_echat/src/com/ibm/lenovo/database/driver/TestDBCreate.java 1.1,ZSHAE,20226
src/rad_echat/webContent/chat/ChatView_01252007.jsp 1.1,ZSHAE,20226
src/rad_echat/webContent/chat/ChatView_12112006.jsp 1.1,ZSHAE,20226
src/rad_echat/webContent/chat/ChatView.jsp 1.1,ZSHAE,20226
```

The format of the exported history version information looks like this:

```
file full path\t version,owner,defect/feature;
```

Figure 5 shows the source history data storage format. Example:

**Figure 5. Exported source code storage format**

```
build.xml@$1.1@$jyoungmn@$19428
```

We are using the following naming convention for all the history data files:

```
file name @$ version @$ owner @$ defect/feature
```

**Import the history data into Rational Team Concert**

**Import users into Rational Team Concert**

Now that we have all of the required data exported, we can start importing the users into Rational Team Concert with the file called User.txt. If there are just a few users, it might be quicker to import them manually. Otherwise, it would be necessary to do that with the Jazz APIs.
Import the defect or feature into Rational Team Concert

Next, we will work on the defect and feature import.

1. Analyze each exported defect or feature data file to get the essential attributes, based on the Defects/Features Mapping rule. You could decide to migrate only fields that you're interested in rather than all the fields. Listing 7 gives the core importing Java code.

Listing 7. The core Java code for importing a defect or feature

```java
private static class WorkItemInitialization extends WorkItemOperation {
    private WorkItemAttributes workItemAttr;
    public WorkItemInitialization(WorkItemAttributes workItemAttr) {
        super("Initializing Work Item");
        this.workItemAttr = workItemAttr;
    }

    @Override
    protected void execute(WorkItemWorkingCopy workingCopy,
                           IProgressMonitor monitor) throws TeamRepositoryException {
        IWorkItem workItem = workingCopy.getWorkItem();
        // set the work item attribute one by one
        // ...
    }
}
// work item creation
WorkItemInitialization operation =
    new WorkItemInitialization(workItemAttribute attr);
IWorkItemHandle handle = operation.run(workItemTypeTask, null);
IWorkItem workItem =
    auditableClient.resolveAuditable(handle, IWorkItem.FULL_PROFILE, null);
int def_fec_id_new = workItem.getId();
// record the return work item id and
// the old CMVC defect/feature name mapping.
sbFeatureMapping.append(def_fec_id + "\t" + def_fec_id_new + "\r\n");
```

2. Call the Rational Team Concert API to create the work item and record the return work item ID. The work item ID is required for importing a specific code version. That is, we will check in each history version with a work item ID that has been created for the feature or defect.

3. After that, write the work item ID and the previous CMVC defect number mapped to a Def_wrkitmID.txt file, as Listing 8 shows.

Listing 8. The defect name and work item ID mapping list (example)

<table>
<thead>
<tr>
<th>ID</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>20278</td>
<td>31</td>
</tr>
<tr>
<td>20279</td>
<td>32</td>
</tr>
<tr>
<td>28280</td>
<td>33</td>
</tr>
</tbody>
</table>

Also, write the work item ID and the previous CMVC feature number, mapped to a Fea_wrkitmID.txt file, as Listing 9 shows.
Listing 9. The feature name and work item ID mapping list

<table>
<thead>
<tr>
<th>CMVC Defect or Feature Number</th>
<th>Rational Team Concert Work Item ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>20226</td>
<td>34</td>
</tr>
<tr>
<td>19428</td>
<td>35</td>
</tr>
<tr>
<td>19167</td>
<td>36</td>
</tr>
<tr>
<td>19405</td>
<td>37</td>
</tr>
<tr>
<td>21318</td>
<td>38</td>
</tr>
<tr>
<td>20286</td>
<td>39</td>
</tr>
</tbody>
</table>

These CMVC defect or feature numbers and Rational Team Concert work item ID mapping relationships will be used while importing the source code. The source code file associated with the defect or feature number in CMVC should be changed to the new work item ID in Rational Team Concert.

**Import Source Code to Rational Team Concert**

We now have the version history information file, Path_historyver.txt. In our version history information, the first part of a file path is the component name, such as “build” and “src.”

1. Create and share the component folders in your Rational Team Concert workspace, as depicted in Figure 6.

**Figure 6. Shared component folders in Rational Team Concert**

2. Import the source code history in the Java program:
   a. Loop each source code file and each history version in the Path_historyver.txt file, and create a change set with the original defect or feature mapping the new work item ID.
   b. Commit the source code to Rational Team Concert workspace, releaseAWorkspace, with the change set, and then deliver the code change to the Rational Team Concert releaseA stream.
   c. While creating the change set, it is a good practice to add CMVC version information in the comment.

Listing 10 presents the core importing Java snippet.
Using the Jazz API to migrate historical data from CMVC to Rational Team Concert projects

Listing 10. The core Java fragment for importing the source file

```java
IFileItem fileItem = null;
if(isNewItem()) {
    fileItem = (IFileItem) IFileItem.ITEM_TYPE.createItem();
} else {
    fileItem = (IFileItem) workspaceConnection.configuration(myComponent)
               .fetchPartialItem(fileItem, null, monitor);
    fileItem = (IFileItem) fileItem.getWorkingCopy();
}

fileItem.setName(fileInfo.fileName);
fileItem.setParent(parentFolder);

FileLineDelimiter delimiter = FileLineDelimiter.LINE_DELIMITER_NONE;

IFileContent storedContent = contentManager.storeContent(
    IFileContent.ENCODING_US_ASCII, delimiter,
    new VersionedContentManagerByteArrayInputStreamPovider(
        readFileToByteArray(fileInfo, version) ),
    null, monitor);

fileItem.setContentType(IFileItem.CONTENT_TYPE_TEXT);
fileItem.setContent(storedContent);
fileItem.setFileTimestamp(new Date());

IChangeSetHandle changeSetHandle =
    workspaceConnection.createChangeSet(myComponent, "--" + version.toString(),
    false, monitor);
fileWorkItemmanager.createLink(workspace, changeSetHandle,
    new IWorkItemHandle[]{handleWorkitem}, monitor);

// commit the change to workspace
workspaceConnection.commit(changeSetHandle,
    Collections.singletonList(workspaceConnection.configurationOpFactory() .save(fileItem)), monitor);

// deliver the changes to the stream
IChangeHistorySyncReport sync =
    workspaceConnection.compareTo(streamConnection,
    WorkspaceComparisonFlags.CHANGE_SET_COMPARISON_ONLY,
    Collections.EMPTY_LIST, monitor);
workspaceConnection.deliver(streamConnection, sync, Collections.EMPTY_LIST,
    sync.outgoingChangeSets(myComponent), monitor);
```

Verify the migration result

Verify the defect or feature

Figure 7 shows the defect/feature migration result for the example in this article.

Figure 7. Migration result
Verify the source code

Figure 8 shows that all of the code files have been imported into Rational Team Concert.

Figure 8. Source files migration result

Verify the source code history

1. Select the build component in the releaseA (Team 1) stream hierarchy, and then right-click and click Show — Repository Files on the drop-down menu.
2. Select the file you'd like to verify (e.g. build.xml of the component build), and then right-click and choose Show History on the drop-down menu. You should be able to see all the history versions.

Figures 9 and 10 show the history version of a source file.

Figure 9. Source file history version migration result, Step 1
Figure 10. Source file history version migration result: Step 2

Now you are able to keep your project assets from the original SCM systems and enjoy the Rational Team Concert benefits.
# Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMVC Export source code</td>
<td>CMVC_Export_src.zip</td>
<td>12KB</td>
</tr>
<tr>
<td>RTC Import source code</td>
<td>RTC_Import_src.zip</td>
<td>10KB</td>
</tr>
</tbody>
</table>
Using the Jazz API to migrate historical data from CMVC to Rational Team Concert projects