WebSphere Business Integration

Message Broker V5

Change Management and Naming Standards

Version 3.0

September 2004

Cyril Stewart

Keith Guttridge

Lakshman Yatawara

Andy Piper

Jonathan Marshall

Emir Garza

IBM UK Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN

Take Note!

Before using this report please read the general information under "Notices".

Second Edition, September 2004.

This edition applies to Version 5 of *WebSphere Business Integration Message Broker* with fixpack 2 (or later).

Notices

Summary of Amendments

Date	Changes
February 2001	Initial version.
June 2004	Updated for WBIMB Version 5.
September 2004	Added Rational ClearCase. Minor editorial changes.

Table of contents

1 Overview

This document is available as IBM SupportPac IC04 from

http://www.ibm.com/software/integration/support/supportpacs/individual/ic04.html

SupportPac IC04 provides:

- Guidelines for Message Broker naming conventions. See *Naming Standards* on page 10.
- Recommendations for version control and change management procedures when using CVS as a repository. See *Change Management with CVS* on page 17.
- Guidelines for organizing your work when not using a repository. See *Change Management* with on page 41.

2 Naming Standards

As a general rule naming standards for MQSeries and Message Broker components should closely follow organizational rules for naming objects but there may be a few subtle differences required.

2.1 Characters allowed in commands

The character set that can be used for naming brokers, execution groups and message identifiers is as follows:

- Uppercase A-Z
- Lowercase a-z
- Numeric 0-9
- Any special characters supported by the underlying file system.

IBM recommends the only non-alphanumeric character used in the name is underscore.

On UNIX platforms, broker names are case sensitive: for example broker names `Broker1` and `BROKER1` refer to different brokers. On Windows this is not the case. For this reason IBM recommends using all uppercase characters to ensure consistency across platforms.

2.2 Converting Version 2.1 names that are not valid

Flows and properties can contain names that are not valid in Message Broker Version 5. If this situation arises, the following transformation occurs. Each offending character is replaced with a series of characters representing its Unicode code point. For example, character "!" is replaced with `X0026`. This is explained in the report file that is generated.

This transformation is deterministic. If a flow is migrated on another occasion, which refers to a flow with a character that is not valid, both names are transformed in the same way.

These transformations do not result in conflicting names except in extremely rare circumstances. A conflict might occur because a Unicode code point sequence occurs in a name precisely where the corresponding character occurs in another name which is otherwise identical. In this case you must rename one of these flows or properties and re-export the flows. You are recommended to select a new name that does not contain a Unicode code point sequence ('`Xnnnn`') and to rename the message flow in the Control Center before you migrate. Never rename a `.msgflow` file in the file system; always use the Control Center or the workbench to perform renaming tasks.

2.3 Message Broker Naming Standards

A naming convention for all components of the Message Broker network is required to ensure that all names are unique and that users have clear guidance as to how to name new objects. This section of the document aims to lay out a suggested naming convention for a Message Broker network.

In IBM WebSphere Business Integration Message Brokers the Control Center has been replaced with WebSphere Message Brokers Toolkit. This gives the build time environment a consistent user interface across the range of WebSphere products. With the introduction of WebSphere Message Brokers Toolkit comes the need to support a naming standard for the new object that are needed when developing message set and flows.

So in addition to the objects that were present in earlier versions of IBM WebSphere Business Integration Message Brokers:

- The Brokers.
- The Execution Groups.
- The Message Flows.
- The message sets.
- The messages.
- The message flow nodes.

There are new object to be included in the naming standard:

- The Message Flow Project.
- The Message Flow ESQL File.
- The Message Flow File.
- The Flow Mapping File.
- The Message Set Project.
- The Message Broker Archive File.

Brokers

The name used at creation time must be unique within the broker domain. Since Queue Manager names are normally unique within a given organization's environment it is suggested that brokers should have the same name as the Queue Managers they are associated with. For example a Queue Manager called `WBIMB` would have a broker `WBIMB` associated with it.

Execution Groups

The name for each Execution Group must be unique within a broker. It would be recommended to separate Execution Groups by department name. This means that all related departmental message flows could be grouped easily within the broker.

There may, however, be a situation where message flows within a department may need to be kept apart from the others (for example if one is particularly large). In this instance the name of the execution group could be the department name with a suffix of the relevant project name or identifier.

One further thing you may wish to consider when working within the development environment is having separate Execution Groups per developer. The benefit of this is that it will avoid interference at deploy time with other developers and also keep each person's work autonomous within system processes for testing purposes. One way of achieving this would be to use a prefix or suffix on the Execution Group name to identify a particular developer. However, you could only use this naming standard in a development environment, as the Execution Group configuration in System Test needs to exactly match that of Production.

Message Sets

The name for a message set must be unique within a broker domain. Any reference to that name within the domain must use that one message set. However, several brokers may use a single message set. A suggested naming standard for message sets could be:

```
ProjectCode_MessageSetDescription
```

1. Project Code is an alphanumeric code representing the project that owns the message set, for example `BillingProject`.

2. Message Set Description is a short description that is concise, clear and understandable as to the function of the message set, for example `AccountingMessages`.

3. All sections of the message set name should be separated by an underscore

Messages

The name for a message must be unique within a message set. Any reference to that name within the message set must use that one message. However, several message sets may use a single message. A suggested naming standard for message could be:

```
ProjectCode_MessageDescription_InputOrOutput
```

1. Project Code is an alphanumeric code representing the project that owns the message, for example `BillingProject`.

2. Message Description is a short description that is concise, clear and understandable as to the function of the message set, for example `AccountValidation`.

3. Input or Output describes whether the message is an input or output message type. If a message is both input and output then this could be set to `InOut` or left blank depending on your preference.

4. All sections of the message name should be separated by an underscore.

Going down into lower levels from messages you get to elements, element lengths etc. Naming standards are normally not needed at this point as element names are normally generated automatically by importing C structures, COBOL copybooks or XML schemas. If you are not using the importer then the message element names should be meaningful.

It should also be noted that message element identifiers will be generated automatically for you and it is recommended to change these to the same names as the elements themselves. The reason for this is that when you use the messages within your message flows you refer to the specific element using the element identifiers and not the element names. Therefore it will make you ESQL easier to write and understand if your element identifiers are the same as your element names.

The importer will automatically generate element lengths. If you have to create these manually then you should give them a meaningful name. For example if you are creating an element length for a string of length 20 the call the element length `Length_20`, or `L20`. This means that you could re-use this component if another message element required the same element length.

Message Flow Nodes

Generally speaking there is no hard and fast way that message flow nodes should be named.

However there are some guidelines that can be presented here in order to make the message flow clearer for people trying to understand its business purpose.

• When naming MQInput and MQOutput nodes always name them the same as the underlying MQSeries queue for clarity.

• When naming Filter nodes give them a question title, for example `Is Name NULL?`

• When naming Compute nodes give them a meaningful name to concisely explain their business purpose.

• If you are using Database nodes try and give them a meaningful name to concisely explain their business purpose.

• When using Publication nodes name the node after the Subscription Point name, or possibly a Business Description of what is being published, or maybe even a combination of both.

The Message Flow Project

The name for a message flow project must be unique within WebSphere Message Brokers Toolkit. It is recommended that the name of the message flow project is related to the name of a project in a version control tool. A suggested naming standard for message flows could be:

```
ProjectCode_MessageFlowDescription_Project
```

If you are using a CVS repository, add a suffix to the project name that identifies the project as a message flow project. For example, append `MF` to the project name. See *Repository Structure* in section *Change Management with CVS*, for more details.

Message Flows (File)

The name for a message flow must be unique within a broker domain. Any reference to that name within the domain must use that one message flow. However, several brokers or execution groups may use a single message flow. A suggested naming standard for message flows could be:

```
ProjectCode_MessageFlowDescription
```

1. Project Code is an alphanumeric code representing the project that owns the message flow, for example BillingProject.

2. Message Flow Description is a short description that is concise, clear and understandable as to the function of the message flow, for example AccountValidation.

3. All sections of the message flow name should be separated by an underscore

The Message Flow ESQL File

The suggested naming standard is to have it's the same as the Message Flow (this is the default behavior).

The Message Set Project

The name for a message set must be unique within a WebSphere Message Brokers Toolkit. The scope of the name does not go beyond WebSphere Message Brokers Toolkit. However, It is recommended that the name of the message set project is related to the name of a project in a version control tool. The recommended standard is:

```
ProjectCode_MessageSetDescription_Project
```

If you are using a CVS repository, add a suffix to the project name that identifies the project as a message set project. For example, append `MSET` to the project name. See *Repository Structure* in section *Change Management with CVS*, for more details.

The Message Definition File

This is the same as a Message Set in version 2.1. Therefore we recommended the same name as that given for the message set project.

The Message Broker Archive File

The Broker archive file (also known as a bar file) is the unit of deployment to the broker. It is recommended that copies of this file are place under version control, when they are promoted through the different environments, such as Integration Test, QA test, and Production.

The recommended naming standard is:

```
ProjectCode_SystemDeployedTo
```

3 Change Management with CVS

This section has 2 purposes:

1. To provide some technical background and recommendations: It outlines some of the basics of System Configuration Management (SCM) usage.

2. To provide a team working framework: It provides an outline of principles of team working with the WebSphere Business Integration Message Brokers Toolkit and an SCM repository – specifically, the Concurrent Versions System (CVS). The concepts are transferable to other source control systems such as Rational ClearCase.

3.1 SCM System Terminology

For those unfamiliar with SCM systems, this section is intended to provide some background to the terminology used. For readers who are familiar with source control systems, *SCM Terminology Comparison* on page19, covers the differences between the Eclipse view of team development, and the terms used by the most popular products (CVS and Rational ClearCase).

3.1.1 Branching

A repository can branch into multiple streams, which are distinguished by name. In a CVS repository, for example, a stream maps to the main trunk HEAD or to a branch. When the same project appears in different streams, the resources evolve independently. Changes released in one stream have no effect on other streams until they are merged.

3.1.2 Commit

After you have caught up with the stream, merged any conflicting changes in your local workbench, and tested all changes locally, you can safely commit your changes to the stream. In this way you can be sure that you are not overwriting work that has been done by other team members.

When you commit your changes to the stream, they will be copied from your local workspace to the stream. As a result, these changes will be seen as incoming changes when other developers later commit to the stream.

3.1.3 Repository

A repository is a persistent store that coordinates multi-user access to projects and team streams. In WebSphere Message Brokers Toolkit, repositories are managed by SCM products such as CVS or Rational ClearCase LT. SCM vendors provide plugins that enable WebSphere Message Brokers Toolkit to communicate with their repository client. Communication between WebSphere Message Brokers Toolkit and the repository server is via TCP/IP. In ClearCase terminology, a repository is a VOB.

3.1.4 Resources

Resources are a collective term for projects, packages, folders, and files that exist in the Workbench. The Navigator view provides a hierarchical view of resources and allows you to open them for editing. Other tools may display and handle these resources differently.

There are three basic types of resources that exist in the Workbench:

Files	Comparable to files as you see them in the file system.
Folders	Comparable to directories on a file system. In the Workbench, folders are contained in projects or other folders. Folders can contain files and other folders.
Projects	Contain folders and files. Projects are used for builds, version management, sharing, and resource organization. Like folders, projects map to directories in the file system. (When you create a project, you specify a location for it in the file system.)

3.1.5 Stream/Branch

A stream is a shared work area that lets team member release their work and use code updates from others. The stream represents the current state of a shared project. A repository can contain multiple streams for different projects or different development stages, such as one stream for new development and another stream for maintenance. In CVS terminology, a stream is a branch.

3.1.6 Synchronizing

When you make changes in the workbench, resources are saved in the local workspace. You need to release resources in order for other team members to access them. At the same time, other team members may have released updates to the stream, which you can access with WebSphere Message Brokers Toolkit. Conflicts arise when you modify a resource and try to release it when there is a more recent version in the stream. In this situation, you can either catch up/update the resource from the stream, release/commit your version of the resource to the stream, or merge your work and the stream resource.

3.1.7 Versions

A version of a file resource captures the content of the file. A version of a project captures the configuration of folders and files as well as their specific versions. You can create new project versions as a snapshot of the current configuration of a project either in a stream, or in the workspace. In the team environment, you cannot explicitly version resources other than projects, such as files or folders. However, they are automatically and implicitly versioned when they are released to the stream.

3.1.8 Update

Update is the process of copying changes other developers have made to a resource into your local workspace. This ensures that you will have the latest work from the other team members incorporated in your work as well.

3.1.9 SCM Terminology Comparison

SCM systems frequently use different terminology to refer to similar SCM commands and functions. The following table lists commands/features and attempts to map them to terminology/features in other SCM systems:

Message Brokers Toolkit	**CVS**	**ClearCase**
Workspace	File System	Work area
Repository	Repository	VOB
Branch	Branch (Tag)	Stream and project
Project	Module	View
Resource	File	Element
Commit	Commit	Check-in
Update	Update	Compare with
Version	Version (Tag)	Version

3.2 CVS Background

CVS is one of many Source Control Management Systems that the WebSphere tools can work with. This document assumes some working knowledge of WebSphere Message Brokers Toolkit and CVS.

- For additional information on using CVS and WebSphere Message Brokers Toolkit, see the online help sections covering CVS.
- For additional information and tutorials on using CVS with the Message Brokers Toolkit, see the Redbook Migration to WebSphere Business Integration Message Broker version 5.

When working with CVS and WebSphere Message Brokers Toolkit, **branches** are the shared storage areas where developers commit the changed resources from their workbench. When a resource is committed to the branch, a new edition of the resource is created. Logical **versions** are created by applying a CVS tag to the collection of current resources in the branch.

The primary model for developing with CVS is to use the default **HEAD** branch, as the branch developers will commit their work to during the standard development cycle. As project builds are required, a new version tag is applied to the contents of the HEAD branch. The project is built with the contents of the version and deployed to test and production environments. When production fixes are required, a **fix branch** is created based on the version currently in production. Code fixes are placed in both the fix branch and the HEAD branch, where ongoing development continues to be done.

3.2.1 Recommended use of multiple branches

It is recommended that development teams do not use the HEAD branch as the development branch. Reasons for this include:

1. **Untrustworthy developers**: The development team should tightly control what code gets committed to the HEAD branch for build purposes.

2. **Interleaved code and build cycles**: This is the case where developers may want to store their code in the SCM system, but it is not ready or scheduled for the next build. The “not ready for prime-time” code may be partially or fully completed, but not scheduled for rollout for a variety of reasons. It is still better to get the code into the SCM system instead of leaving it on the developers’ workstations. In some cases, special branches should be created for significant long-term efforts, but in other cases a single development branch can be used to collect the changes from different developers that are not ready for inclusion in the HEAD branch.

3. **Frequent Production Builds**: In those development environments where small changes are frequently pushed out into production, untested changes need to be kept separated from CVS branch.

3.3 Team working principles

1. Development must always be done on branches

2. Production level code should always reside in the HEAD stream

3. Developers should only commit changes to the repository once testing has ensured that it does not break the existing code.

4. Work can only be performed on a branch's current contents. Do not start development on back version levels: you will not be able to commit changes. Instead first create branch on that version before progressing.

5. Versions can be used for:

    a. Deployment level.

    b. Branching reference.

    c. 'Comparing with' to retrieve back levels of code into current stream.

3.4 Version conventions

Note: periods and spaces are not permitted in CVS version names.

Naming conventions must be decided on prior to development and should include the following:

1. Versioning policy.
   It should be clear how the version number should be incremented every time code is versioned. For example, if the first production version is `V1-0-0`:
   Increment to `V1-0-1` on a small fix, to `V1-1-0` on a feature release, and to `V2-0-0` on a new version.

2. Branch name.
   Each branch and category of branch should have a naming convention. For example:

   a. BASELINE

   b. MAINTENANCE

   c. CRxxx

3.5 Repository Structure

There are essentially two approaches that can be taken to structuring the repository:

1. Flat, i.e. all projects at the top-level of a CVS repository root. This is the structure which is probably easiest to configure when working with CVS and WebSphere Message Brokers Toolkit. This is because a single CVS repository root is used. Projects are added to the workspace on an individual basis, or on a project set basis. The downside is that unless projects are consistently named, the organization of the repository can become unstructured.

2. Multiple roots. If desired, multiple CVS repository roots can be configured, for example to accommodate multiple sub-organizations or different types of development (Message Broker and Java). Since WebSphere Message Brokers Toolkit can connect to multiple repositories, this is also perfectly workable since projects from different sources can be added to the workspace.

Most development environments are not likely to be sufficiently complicated to require multiple repository roots at first. If projects come along later which wish to segregate their development resources, CVS can be configured to provide an additional repository root for that project.

The recommended initial approach is for a flat structure: all projects at the top-level of a single CVS repository root.

Project sets should be used to relate different development resources together.

All project set files, and any common configuration files will reside in a common CVS module called ***ApplicationResources***.

CVS has no concept of project “types”. However, WebSphere Message Brokers Toolkit relies on “typing” a project to know how to treat it in development terms. Therefore it is useful to add an indicator to the project name to enable project types to be quickly identified when looking at the source control system.

Message Flow projects and Message Set projects will be named appropriately to include the business Project name as a prefix, and the WebSphere Message Brokers Toolkit project type as a suffix.

Modules containing shareable code (common ESQL routines, etc.) will have a prefix of `Common`:

> `XYZTransferMF`: a message flow project belonging to the XYZ business project.
>
> `CommonConversionMF`: a message flow project containing shared ESQL library code.

Note that spaces are not valid in CVS module names. Use underscores (_) instead if you need to use a space, but in general this should be avoided.

3.6 Initial Team Working Configuration

Here we outline the steps that need to be taken to provide the relevant resources and team working practices:

3.6.1 Set up repository

Create a CVS repository on a server accessible to all. This is recommended to be a UNIX or Linux server, because the CVS server on Windows is not officially supported with WebSphere Message Brokers Toolkit[1].

Establish a regular back up of the repository.

3.6.2 Create user ids

Each developer working as part of the development team will need a user id set up on the OS of the CVS server. This will be the user id that they use to connect to CVS and all changes will be logged against this username.

3.6.3 Create baseline version

Once the baseline code base is established in WebSphere Message Brokers Toolkit, share the contents into CVS. Tag this code as the Initial Code drop in the new repository.

[1] This is due to stability problems. For more information see http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/platform-vcm-home/docs/online/cvs_features2.0/cvs-faq.html.

However, some customers have successfully run CVSNT v2.0.x on Windows.

3.6.4 File type settings

The Message Brokers Toolkit does not come pre-configured with correct file type settings for use with CVS. When working with CVS, files can be added to the repository as ASCII or binary. By default, the Toolkit assumes that unknown files are binary, unless the file extensions are configured to be recognized as ASCII. Most of the files created when working with message flows contain ASCII data. Before attempting to add any files to the repository, the Toolkit needs to be configured to tell CVS about the contents of all of the different file types. Do this via the `Window → Preferences → Team → File Contents` menu option. Configure files types as shown in Table 1.

Table 1 File extensions used by the Message Brokers Toolkit

Extension	Content	Description
esql	ASCII	Message flow ESQL file
msgflow	ASCII	Message flow file
mfmap	ASCII	Message flow mapping file
mset	ASCII	Message set file
mxsd	ASCII	Message definition file
category	ASCII	Message category file
msgnode	ASCII	Message flow node file
xmi	ASCII	Artifact definition file
configmgr	ASCII	Domain connection file
bar	Binary	Broker archive file
enqueue	ASCII	Enqueue message file

3.6.5 Create project set

A project set is an XML file that lists the projects required as part of the development project.

Create the project set including all of the development projects:

1. Message Flow projects.
2. Message Set projects.

This is done as follows:

1. Load workspace with projects from the correct branches that are to comprise the project set.
2. Select `Export → Team Project Set` and select the destination for this file. Name it according to the branch that it represents. Place this file in the ApplicationResources project, which should also be in CVS.

3.6.6 Procedure for creating workspace

The procedure to create the WebSphere Message Brokers Toolkit workspace for each developer is as follows:

1. Connect to repository.
   Switch to the CVS Repository Exploring perspective
   Create the new repository connecting to the CVS server and supplying the developer's userid and password.

2. Set up workspace variables.
   This can be useful if an environment independent path is required.

3. Import project set.
   Import the project set file. This will configure the workspace with the latest committed contents of the application projects.

3.7 Team work procedure

This section provides a framework for team development with WebSphere Message Brokers Toolkit and CVS.

A brief outline of the team working principles are mentioned here, but further information on this can be found in:

- A Team Development tutorial on the WebSphere Developer Domain (see References on page 40) .
- The WebSphere Message Brokers Toolkit Help System, under *Team programming with CVS*.

3.7.1 Using CVS during Development

When a developer is working on code, he should be working in a WebSphere Message Brokers Toolkit workspace that has projects associated with the **primary development branch**. The standard developer should never commit items directly to the HEAD branch as the HEAD branch is used for integration and building of releases.

The development cycle

The standard development cycle is as follows:

- Set up the development branch. When the projects are initially shared, they should be added and committed to the HEAD branch. At this point, the new development branch should be created from the HEAD branch. For the purposes of this document, this branch is known as DevBranch
- The developer initializes his workbench by connecting to CVS and pulling the projects from the DevBranch branch. It should only be necessary to do this once.
- Prior to starting a new development effort, the developer will refresh the workspace with the latest code from the development branch. This can be done be selecting all of the projects and use either the Team…Synchronize option or the Replace…Latest from Repository option.
- Fully build and test code prior to committing the code back to the development branch. Check in code as follows:
    1. Select the projects and do a Team…Synchronize with Repository… action.
    2. From the Synchronize view, analyze the Incoming Mode first. Identify any changes that were placed to the repository since his last synchronization. The Show Conflicts page will highlight the incoming changes that directly conflict with the outgoing changes. Any changes should be brought into his workbench, through either direct replacement or merging of code. Any conflicts with the developer’s own modifications should be resolved. The code should then be retested.

3. After processing incoming changes, resynchronize (if necessary) and commit outgoing changes using the Outgoing mode.

New Projects

When a new project is created and associated with CVS with the **Team…Share Project** menu action, the project will be associated with the HEAD branch. Before synchronizing with the repository, use the **Team…Branch** menu action to move the project over to the development branch.

Although the dialog refers to a "new branch", an existing branch can be specified. In this case, the developer should specify **DevBranch** as the branch name.

New Branches

Although branches are an easy way of isolating code from the main development stream for many reasons, branching should not be conducted outside the strict guidelines provided in this document.

If code does need to be shared between developers but not merged into the main development stream, then the developers should consider using the patch capability of WebSphere Message Brokers Toolkit. However, patching can be difficult when using resources generated by the Message Brokers Toolkit because the files are visually-edited but XML-based, making standard `diff` comparisons of questionable value.

Versioning Outline

The following outlines the principles involved in versioning. This discusses the simple scenario, where the main code base resides purely in the HEAD stream. This can be extrapolated into the situation where parallel development is taking place on multiple branches.

There are two approaches to versioning.

Standard versioning

1. Load HEAD contents into workspace
2. Merge in changes from DevBranch
3. Synchronize and commit changes back into HEAD stream
4. Tag as version. After the branch has been versioned, the application should be rebuilt, tested, and deployed.

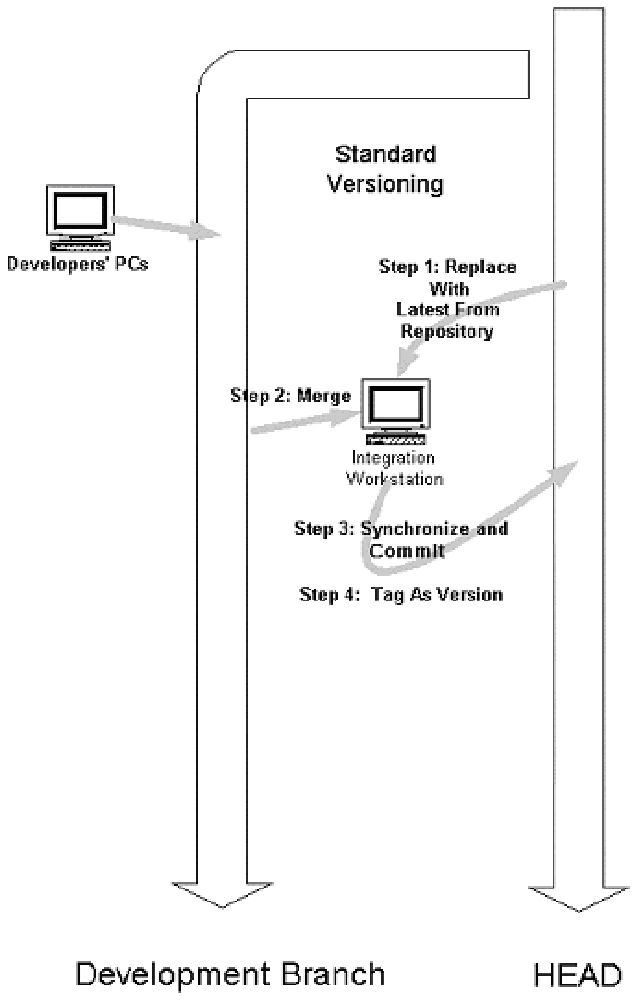


Figure 1. An overview of CVS versioning process

Fix versioning

These are the steps to create and release a fix using a Fix Branch. (See Figure 2)

1. The new fix branch must be created from a previous version of the HEAD branch. Determine the version of the code that is in production and use this version tag as the basis of the fix branch.

2. Load contents of new fix branch into workspace.

3. Fix problem and test.

4. Synchronize and commit changes to fix branch.

5. Merge fix into DevBranch (or notify developers of change).
   The fix will typically be merged into HEAD as well, so that the fix can be rolled into production quickly.

6. The integrator should load the current fix branch contents, which should include the fix.

7. Tag as version. After the branch has been versioned, the application should be rebuilt, tested, and deployed.

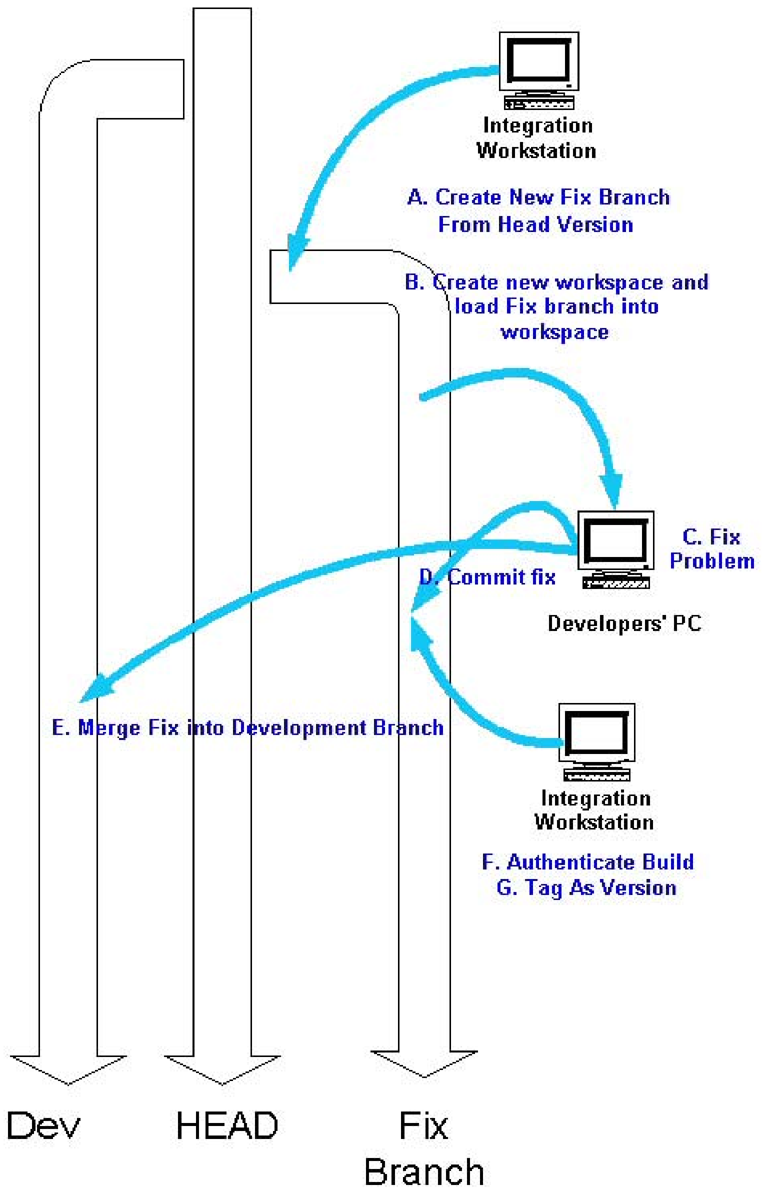


Figure 2. Versioning from a Fix Branch

Versioning with multiple branches

As mentioned in the previous section, versioning can be extended to work on multiple branches. The following is an outline of how this can work

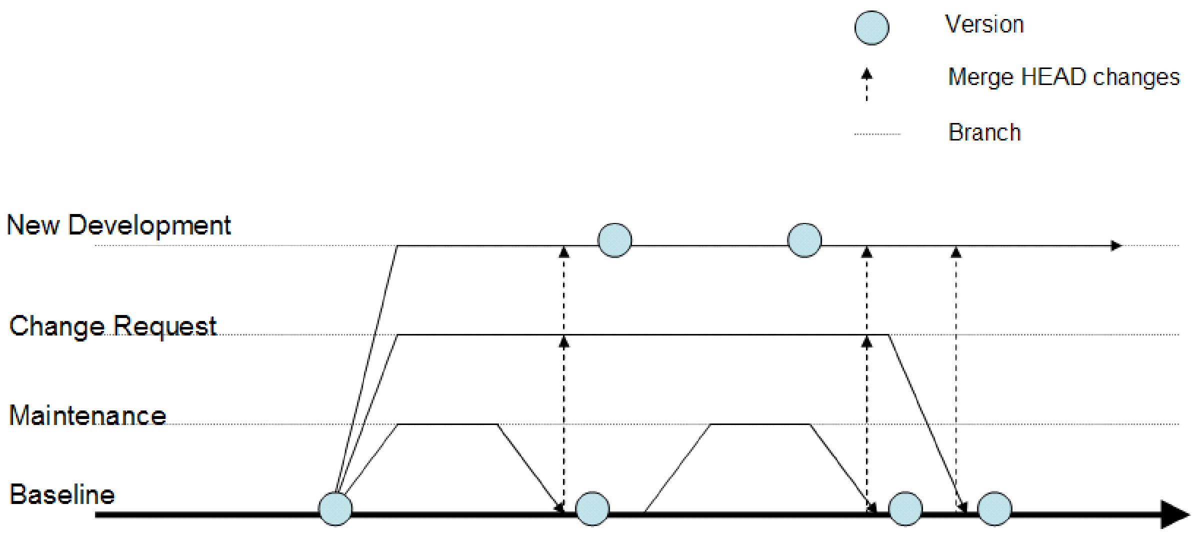


Figure 3 Versioning with multiple branches

Branch usage

BASELINE (HEAD stream) - This will contain the production ready code at all times. It is the responsibility of a designated integrator to merge changes into this branch.

MAINTENANCE - This branch will contain maintenance fixes. Although mentioned here as one branch, that is only conceptual. A branch will be created per fix, tested and then merged back into the BASELINE. (Hence the importance of naming conventions: all MAINTENANCE branches should be prefixed with MAINT.)

CHANGE REQUEST (CR) – Similarly this is a branch created for a single change request. It is expected to by longer living that a maintenance fix, but will not require versioning as part of the branch life cycle. It will have to cater for maintenance fixes being made and will need them to be manually merged into this branch.

NEW DEVELOPMENT – This branch will hold the new development code. It will need to merge in changes made in the CR and MAINT branches. This branch will need periodic versioning at milestones in the development.

Version usage

As previously mentioned, a version can not be further developed. A version is a means of marking a milestone in the code development.

The CVS term for a version is a 'tag'. A tag is a way of labeling together all the resources in a project at a specific resource revision level. For example:

`TestProject` is tagged as V1_0 and contains

`Test.msgflow`	at 1.1
`Test.esql`	at 1.2.1

There are 2 options that can be used to perform branching and merging in WebSphere Message Brokers Toolkit.

1. Manually branch (not recommended)
    a. Replace workspace contents with a previous version of a project
    b. Create new branch
    c. Make changes
    d. Synchronize and commit (which will commit the changes back into the new branch)
2. Merge changes back into current branch contents (recommended)
   If old code needs to be retrieved for further development. The code should be retrieved from the required version and merged back into the current contents of the working branch. A simple **Compare with…version** should suffice.

To continue development, the latest contents of a branch (including HEAD) must be loaded into the workspace.

A version provides functionality for:

a. Deployment – A specific level of code can deployed to production
b. Branching reference – When a branch is created, it needs to be 'rooted' in a given version of the stream that it is branching from.
c. 'Comparing with' to retrieve back levels of code into current stream

Inserting version information in message flows and bar files

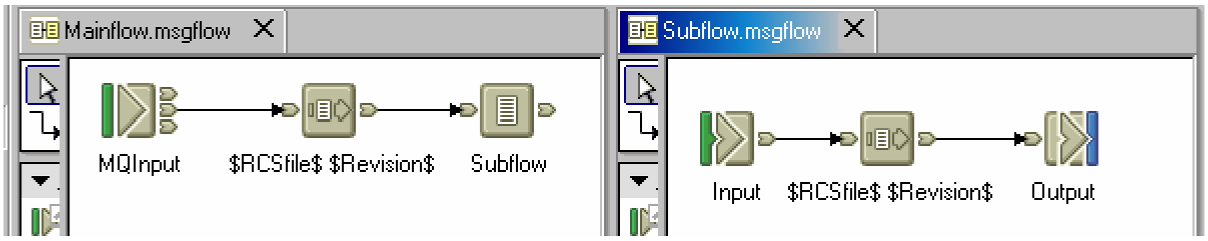
Most version control systems, including CVS, support the concept of "tags" which are special character sequences in text files. These tags can be automatically replaced by the extract command with the revision value of the corresponding data in the repository.

More detailed information is available at
http://www.cvshome.org/docs/manual/cvs-1.11.7/cvs_12.html#SEC98.

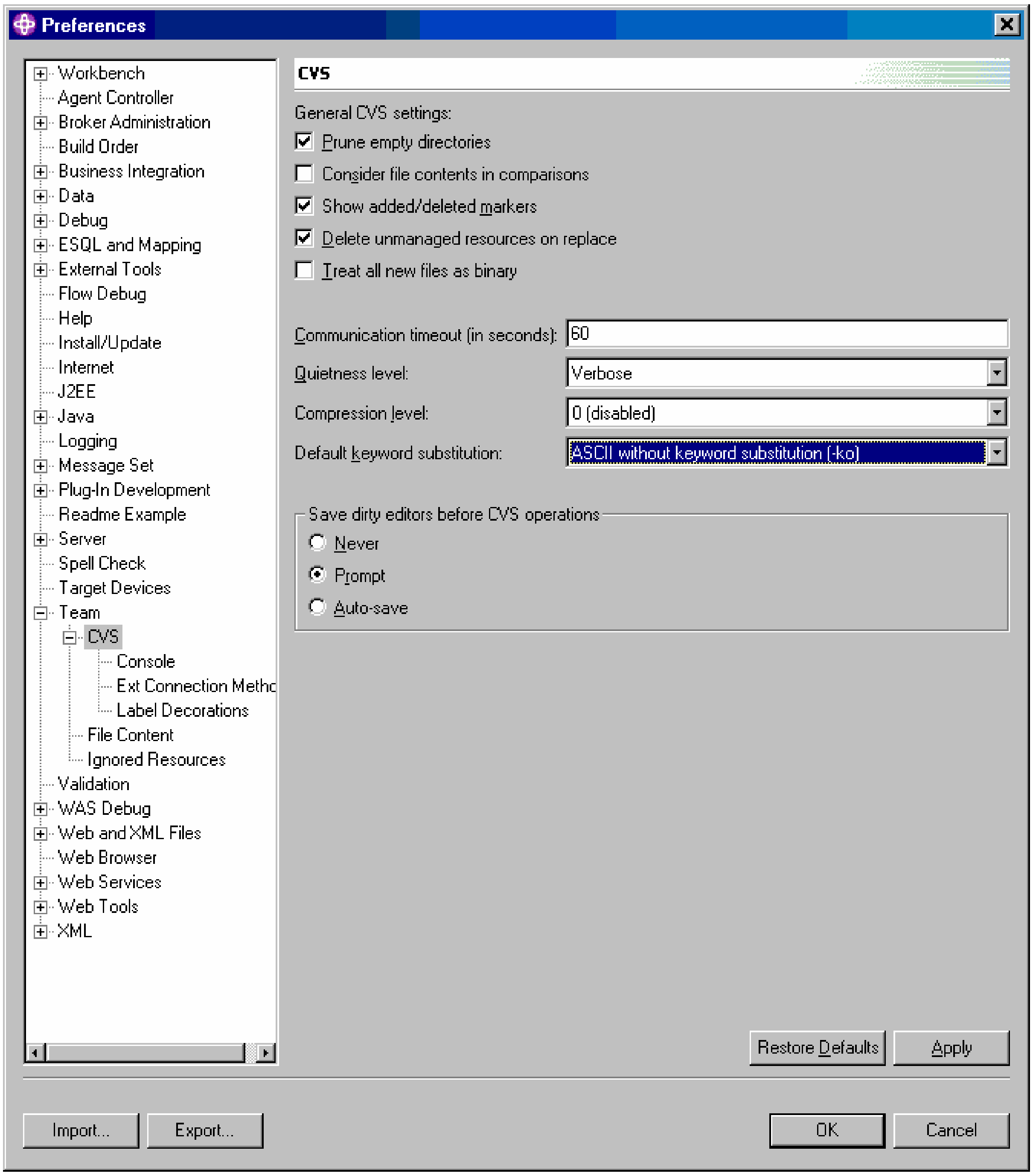
First, add a Passthru node to the subflow. Name it with the key **$RCSfile$ $Revision$**. Nodes are permitted to have pretty much any name at all, so this is acceptable to the flow compiler. However, avoid characters that will be XML escaped: the encoded version will be different from the edited version!

Notice that the file name and version number do not have to be maintained by the user. The Passthru node was added to v5.0 Fixpack 2 precisely to support this scenario.

The following example contains two flows: "Mainflow.msgflow" and "Subflow.msgflow". The first node in each flow is a Passthru node with the name set to **$RCSfile$ $Revision$**. **$RCSFile$** is the name of the file in CVS, and **$Revision$** is the file revision number (of the form X.Y[.x.y]). Additionally, the **$Name$** tag can be used to substitute the version name being extracted.



By default, the CVS client in Eclipse stores msgflow files as ASCII without keyword substitution. See **Window>Preferences...**; then select **Team>CVS** to see the current value. CVS can auto-expand the tags on commit, but this example uses the **ASCII without keyword substitution** mode so they will produce nicer labels when editing the file. CVS supports several options for substituting the tag; choose the one that best suits your needs..


Preferences

- Workbench
- Agent Controller
- Broker Administration
- Build Order
- Business Integration
- Data
- Debug
- ESQL and Mapping
- External Tools
- Flow Debug
- Help
- Install/Update
- Internet
- J2EE
- Java
- Logging
- Message Set
- Plug-In Development
- Readme Example
- Server
- Spell Check
- Target Devices
- Team
  - CVS
    - Console
    - Ext Connection Metho
    - Label Decorations
  - File Content
  - Ignored Resources
- Validation
- WAS Debug
- Web and XML Files
- Web Browser
- Web Services
- Web Tools
- XML

CVS

General CVS settings:

- [✓] Prune empty directories
- [ ] Consider file contents in comparisons
- [✓] Show added/deleted markers
- [✓] Delete unmanaged resources on replace
- [ ] Treat all new files as binary

Communication timeout (in seconds):	60
Quietness level:	Verbose
Compression level:	0 (disabled)
Default keyword substitution:	ASCII without keyword substitution (-ko)

Save dirty editors before CVS operations

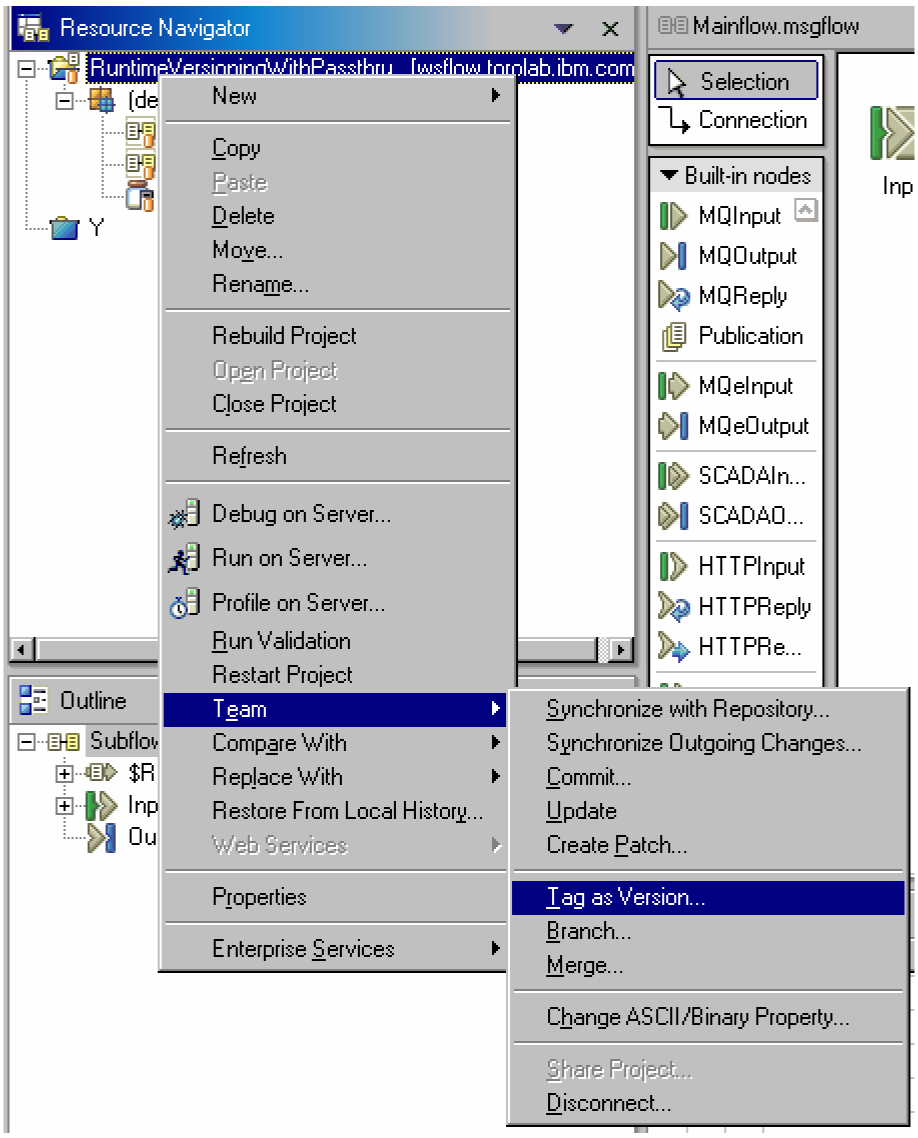
Never [✓ Prompt] Auto-save

Commit the flows to the CVS repository using the **Team>Synchronize with Repository...** menu. This completes instumenting the flow to contain file revision numbers.

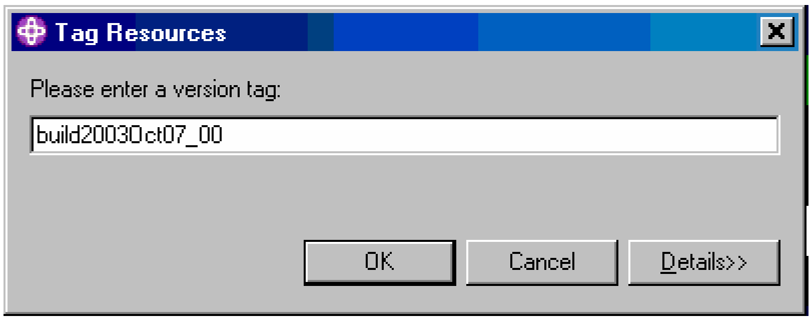
Next, the deployer (the human administrator) must specify the build version tag, extract the flows from the CVS repository and build the broker archive file. All these steps can be performed in a single repeatable script using console tools.

The Eclipse CVS client or the CVS console tools can be used to tag the build id. The console tools allow for automating the tagging as part of a script. This example uses the Eclipse CVS client.

First, select the project and click the **Team>Tag as Version...** menu.



Next, specify the version tag. CVS applies this tag to all the most recent file revisions committed to the repository in the current branch.



Please enter a version tag: build2003Oct07_00

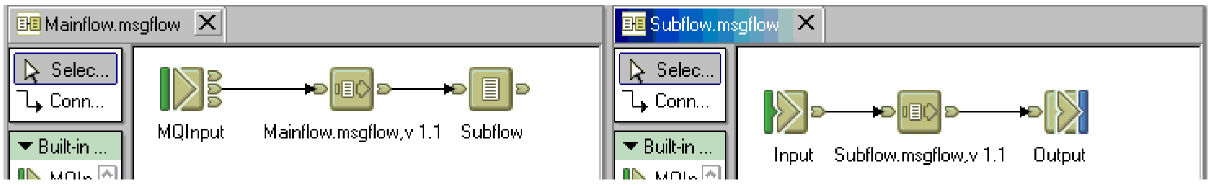
This completes tagging the files with the build version number.

Next, export the build version from the repository. Use the CVS **export** command with the **-kv** option to replace the tag with its value.

```
cvs export -r build2003Oct07_00 -kv
RuntimeVersioningWithPassthru
```

This extracts the file and replaces the **$...$** CVS keywords with their current values, using the current directory as the repository. The **-d *dir*** CVS option can be used to specify a different repository location.

Open the files in the flow editor to see what they look like after extracting with **cvs extract**. The Passthru node names are updated to contain the file and revision number in the source control repository.



This completes naming the Passthru node with the repository tags.

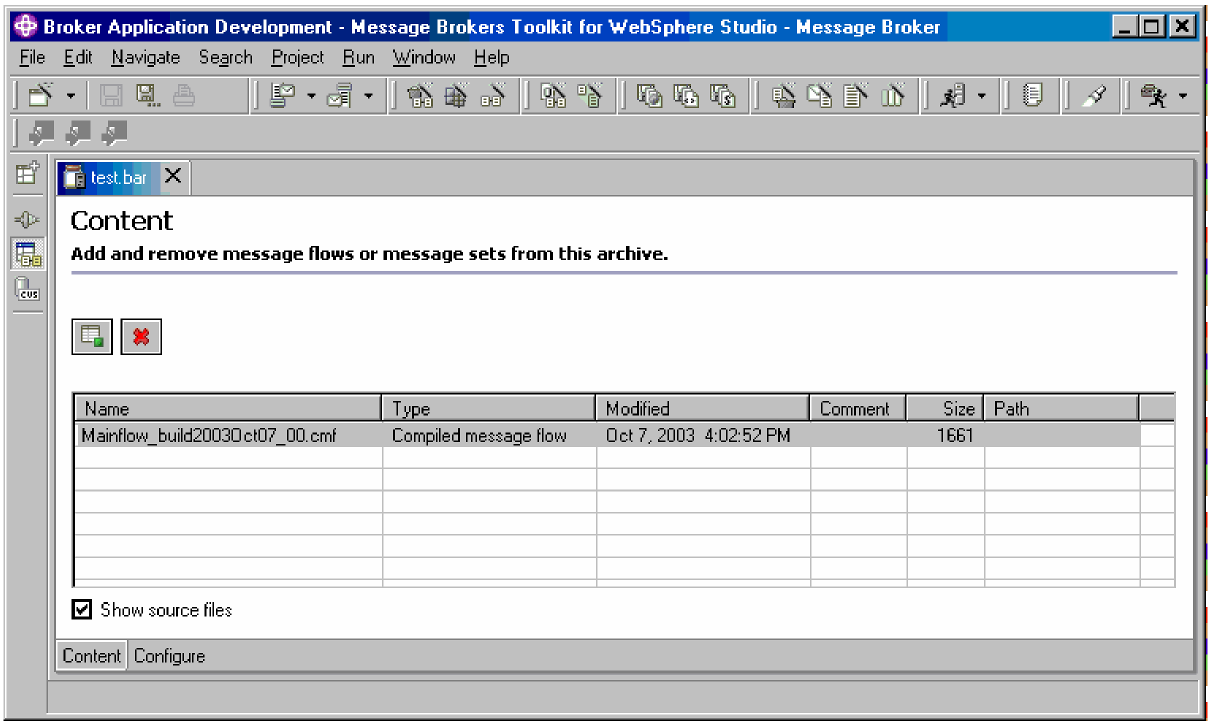
Adding and deploying these flows to a bar file will result in the Passthru nodes logging their name, in other words the file name and revision number, in the trace logs. The **mqsicreatebar** command has a **-v** option to specify the build version number in the flow. For example, building these flows with:

```
mqsicreatebar -b test.bar -version build2003Oct07_00 -o
RuntimeVersioningWithPassthru\Mainflow.msgflow
```

This will create the following compiled flow in the bar file:

```
Mainflow_build2003Oct07_00.cmf
```

Here is the resulting bar file loaded in the Broker Archive Editor


Broker Application Development - Message Brokers Toolkit for WebSphere Studio - Message Broker
File Edit Navigate Search Project Run Window Help
test.bar

Content

Add and remove message flows or message sets from this archive.

Name	Type	Modified	Comment	Size	Path
Mainflow_build2003Oct07_00.cmf	Compiled message flow	Oct 7, 2003 4:02:52 PM		1661	

☑ Show source files

This flow has the build version (**build2003Oct07_00**) in the message flow name and the file revision for each file added to the trace log by the Passthru node.

3.8 References

WebSphere Message Brokers Toolkit Help – see “Team Programming with CVS”

IBM Website: developerWorks
http://www.ibm.com/developerWorks

Migration to WebSphere Business Integration Message Broker v5

http://www.redbooks.ibm.com book no: SG24-6995

4 Change Management with Rational ClearCase

4.1 Change Management with ClearCase

The purpose of this section is to give the reader an understanding of Software Configuration Management (SCM) and how ClearCase addresses SCM issues. Before going into the details it is best to see the terms ClearCase uses to aid SCM. Rational have also developed something called Unified Change Management (UCM). This is an out of the box process to give best practices for SCM. It allows for a wider control of elements and ties them into components, projects and activities. The descriptions below are from the web-based course "Getting Started with Rational ClearCase: Concepts and Terminology for WebSphere Studio v5 Users"[2].

4.1.1 Basic Terms

SCM – Software Configuration Management: Software Configuration Management is the software engineering discipline that deals with managing change to software. The purpose of the discipline is to ensure that each stage of the software development process is controllable and reproducible. This means that all aspects of the project can be readily identified and managed and that any stage of the project can be easily recreated.

ClearCase: Rational ClearCase is an SCM system used to control multiple versions of files used in software development projects. ClearCase is the enterprise level solution designed to handle multiple teams working in parallel across several locations if required.

ClearCase LT: Rational ClearCase LT is a lightweight version of ClearCase designed for smaller development projects. It contains a subset of the full ClearCase implementation.

ClearQuest: Rational ClearQuest is a defect and change tracking application designed to aid companies in managing and tracking change together with reasons for the change etc.

4.1.2 ClearCase SCM and UCM Terminology

Element: An element is a file or directory under ClearCase control. Files and directories become elements when they are checked into ClearCase.

Version: A version is a revision of an element that is being managed by ClearCase. Each time an element is checked in after change a new version of that element is created. The history of these changes is known as version trees.

VOB – Versioned Object Database: A VOB is a data repository used by ClearCase for storing versions of elements and the metadata associated with these objects.

Checkout: To modify an element you must first check it out. This allows you to modify the element in your view.

[2] http://www.ibm.com/developerworks/rational/library/4195.html

Checkin: Once you have finished modifying the element you must check it in. This creates a new version of the element in the VOB.

View: A view provides a way of seeing a version of each element in such a way that you can modify elements, compile source for testing and create documents. There are two types of view, snapshot and dynamic.

Dynamic View: A dynamic view is one that uses the ClearCase Multi-Version File System (MVFS) to provide immediate, transparent access to elements stored in the VOB. This view is only available in the full ClearCase product.

Snapshot View: A snapshot view copies the required files from the VOB to a local directory on your machine. This view is needed if you wish to be able to compile components without being connected to the network.

Serial Development: The process in which different parts of a software development project are created/modified by a single team serially, ie. Phase 1 completed before phase 2 can be started.

Parallel Development: The process in which different parts of a software development project are created/modified by multiple teams, in parallel. The various parts are then combined into a single integration area.

Merge: Merge is the process involving the integration of two or more versions of an element into a final version.

Development Workspace: The development workspace is an isolated work environment for developers. A project can have multiple developer workspaces which means that a developer may also have multiple workspaces. A development workspace is the combination of a development stream and a development workspace

Component: A component is an item stored in a VOB that groups together a set of directory and file elements within a UCM project. The elements of a component are usually developed and released at the same time.

Project: A project is an item stored in a VOB containing configuration information for a large development effort such as a product release. A project must contain at least one and usually many components. A typical project will contain one integration stream and many development streams.

PVOB (Project VOB): A VOB that stores UCM items such as projects, stream, activities and change sets. Every UCM must have a PVOB. Multiple projects can share the same PVOB but ClearCase components cannot cross PVOB boundaries.

Baseline: A baseline is an item stored in a VOB that typically represents a stable configuration of one or more components. A baseline identifies activities and one version of every element visible in each component.

Activity: An activity is and item stored in a VOB that tracks the work required to complete a development task. An activity includes a headline to describe the task and a change set to identify all the elements the developer creates or modifies whilst working on the activity.

Change Set: A change set is a list of related versions associated with a UCM activity. ClearCase records the versions used by the developer whilst working on an activity.

Stream: A stream connects an activity.

Development Stream: A development stream is an item stored in a VOB that determines what versions of elements appear in a development view. Development streams also maintain a list of developer activities.

Development View: A development view is a view associated with a development stream. The view uses the development streams configuration to select the correct version of the components to work on.

Integration Workspace: An integration workspace is an area where changes made by developers working in isolation are merged. The integration stream and corresponding view represent the projects primary shared workspace.

Integration Stream: A project contains one integration stream that records the projects baseline and enables access to versions of the projects shared elements.

Integration View: The integration view is associated with the integration stream and uses the streams configuration to select the correct versions from the component to work on.

Deliver: The process of merging activities from one stream to another, typically from the development view to the integration stream.

Rebase: Rebase is a ClearCase operation that makes a development work area current with the set of versions represented by a more recent baseline in another stream, usually the projects integration stream or a feature specific development stream.

4.2 ClearCase Overview

Rational ClearCase is a product to utilise SCM in software development. It is available as Rational ClearCase LT, Rational ClearCase and rational ClearCase Multisite.

ClearCase uses a reserved access protocol for managing access to elements. This puts locks in place to prevent concurrent changes to the same element. ClearCase does support a more relaxed access protocol using unreserved checkouts although it is not recommended in most cases.

ClearCase provides lots of tools wizards to help set up and manage the various versioned object bases (VOBs), views, jobs and logs. The ClearCase explorer client can be used to create and work with views. Reporting tools are also provided.

ClearCase relies on the operating system for its authentication, which means that a Microsoft Windows Domain will be required when the server is on a Windows machine and there are multiple users.

The smallest part of ClearCase is the element. An element can be either a directory or a file. ClearCase manages the changes to these elements

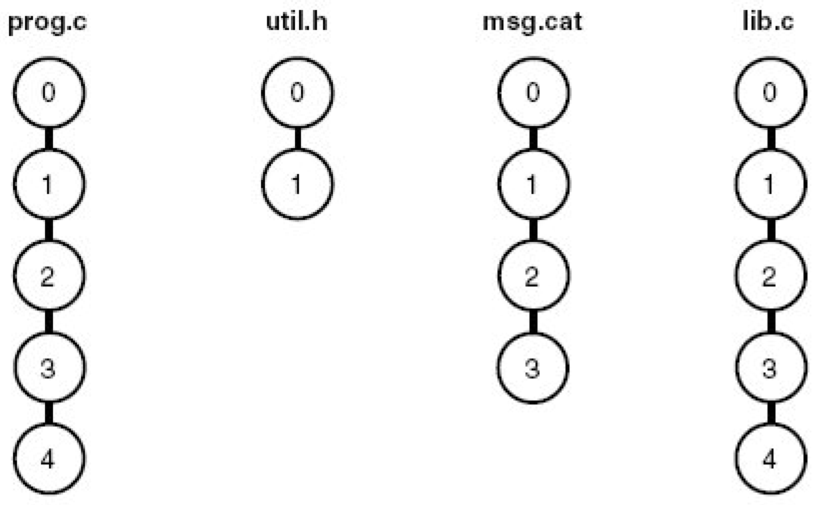


Figure 4: Elements and versions

The diagram in figure 1 shows four elements and the sequence of versions for each element. All of the elements are stored in the VOB. A ClearCase server can have multiple VOBs, but has to have at least one.

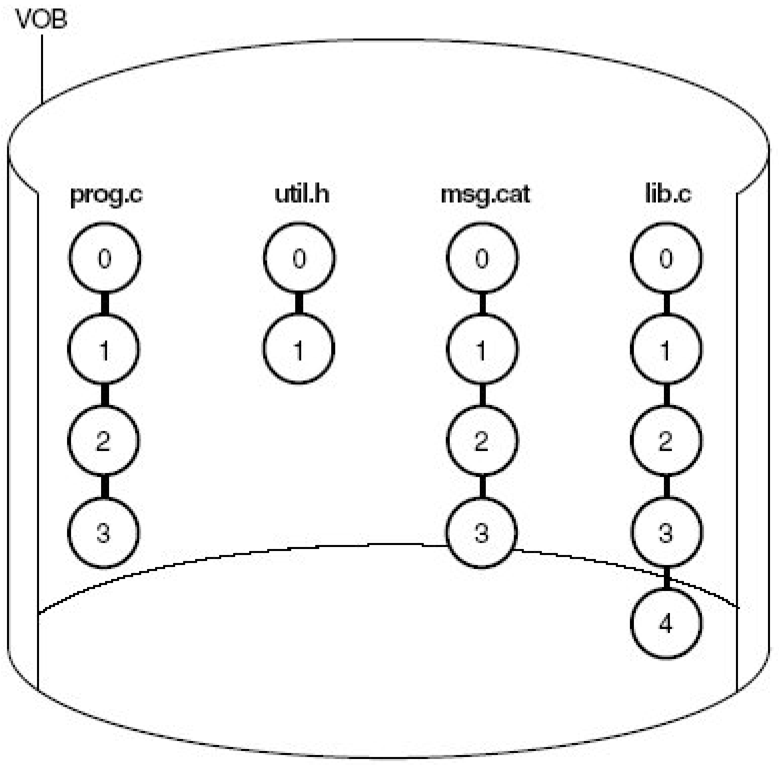


Figure 5: VOB

The VOB is created using the VOB Creation Wizard, which can be launched from the start menu or from within the ClearCase Administration Console.

Once the VOB is created you need to create a view to access the elements in the VOB. Each VOB can have multiple views. Typically there will be one view for each developer and on larger projects, one view for the integration stream. To create the view you use the View Creation Wizard which again can be launched from either the start menu or from within the ClearCase Administration Console.

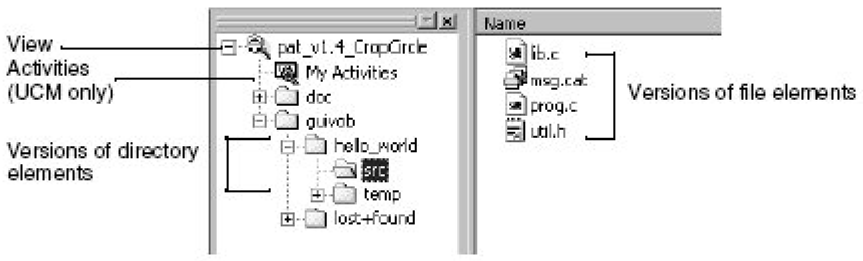


Figure 6: View

Once the view is created we are ready to start populating the VOB. The simplest way to do this is using the ClearCase Explorer. Simply add the new elements into the VOB and select Add to Source Control from the context menu. This allows you to add a comment to explain why the elements are added.

To make changes to elements in the VOB you must first check them out. Once checked out you may perform any changes you wish to make. To confirm the changes made you must check in the element. If you do not wish to make any changes you can simply select Undo Checkout from the context menu instead. Remember to enter a comment for each change made.

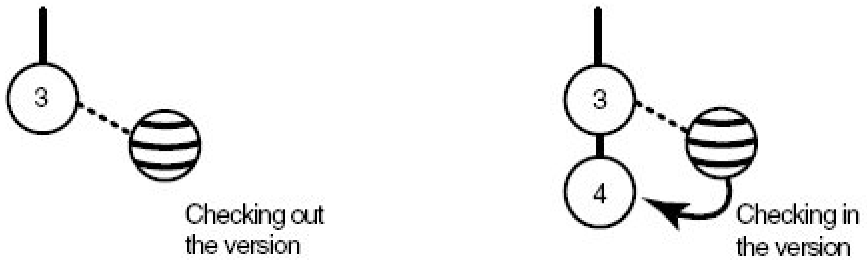


Figure 7 : Check in Check out process

If other people are also working on the project then you may need to retrieve their changes as well. To do this you select the Update View from the context menu. This will ensure all changed files are retrieved allowing the development to progress with the correct level of files.

When we have the project at a reasonable state we will want to take a baseline. You can think of the baseline as a first draft. A baseline identifies one version of each element in the view.

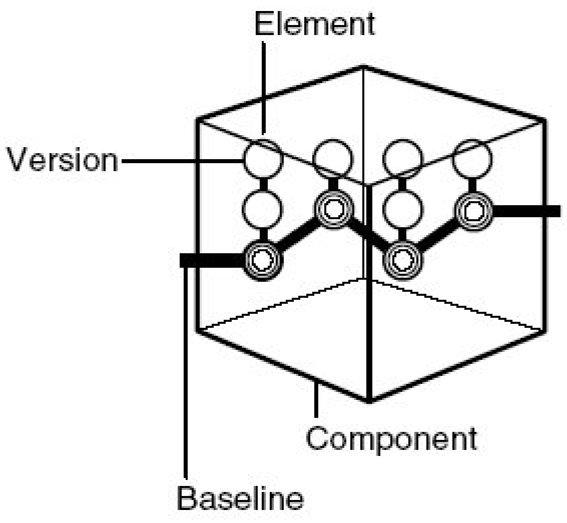


Figure 8: Baseline

There will be several baselines, one for each stream. Typically there will be an integration stream and several development streams. A stream links together views, activities and baselines.

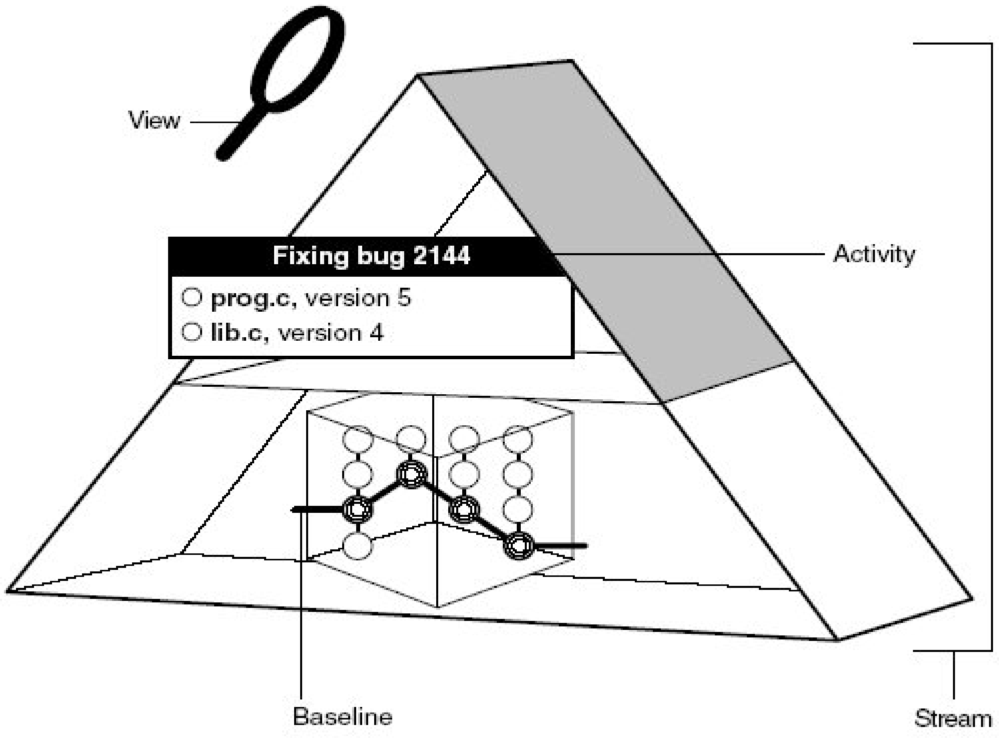


Figure 9 : Stream

Each developers stream can be considered a private work area so multiple streams will exist.

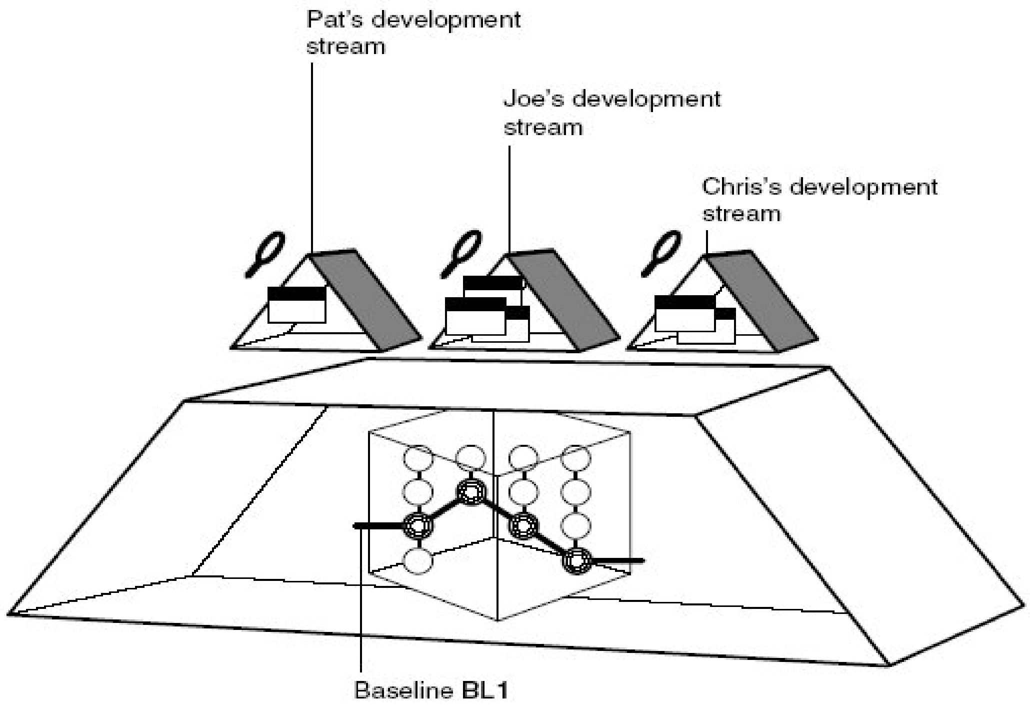


Figure 10 : Private work areas

Once the developer has finished working with their stream and wish to merge it into the integration stream they must deliver their stream.

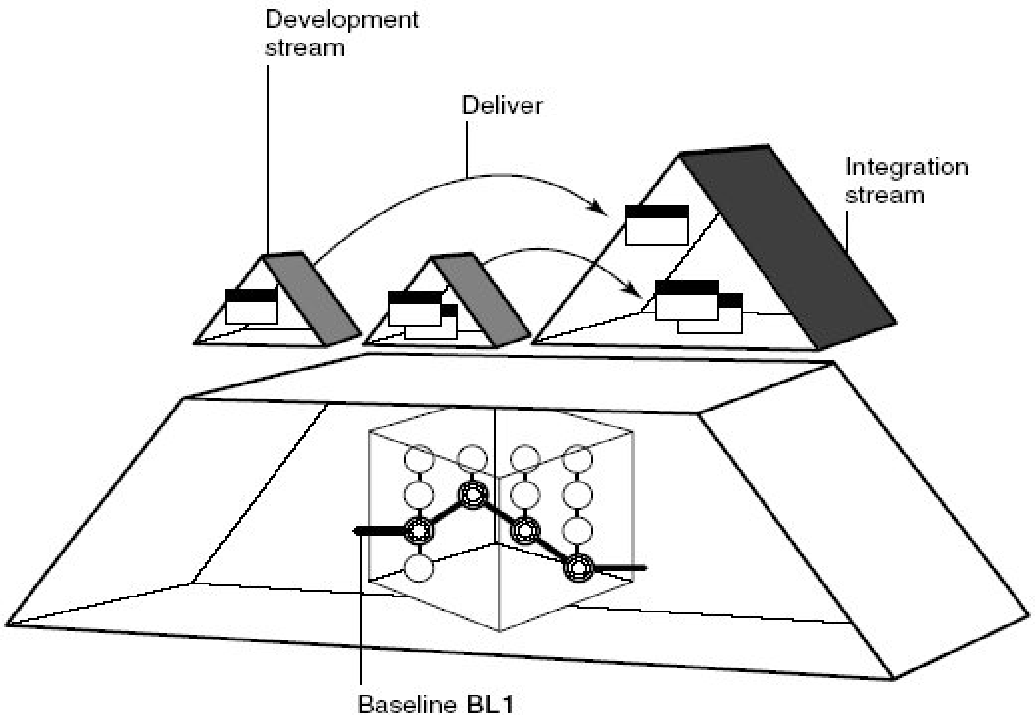


Figure 11: Delivering streams

Every now and then at important milestones, a new baseline will be created in the integration stream. This will include all of the changes delivered since the last baseline was taken. At this point the developers may wish to update the baselines in their stream. This is known as rebasing the development stream.

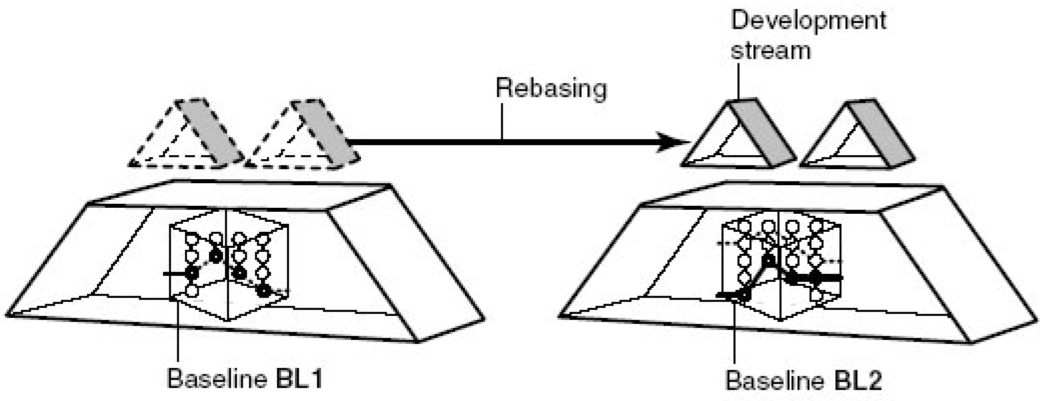


Figure 12: Rebase

After a while the project will be in a stable state, for example, when reaching a release level. At this point you may wish to take a version. A version is actually a label applied to the current versions of all the elements in the view.

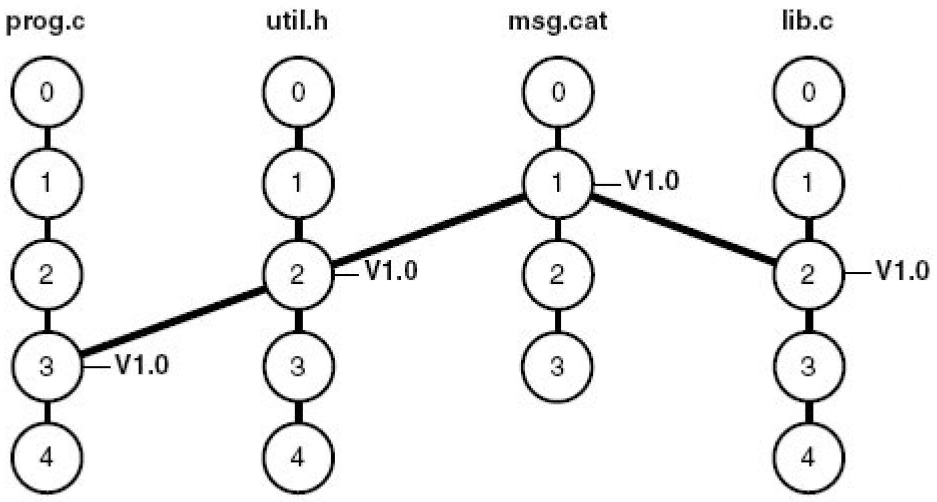


Figure 13: Version

4.3 Working with Branches

When working with multiple developers we will use multiple branches of elements. Imagine we have a project. It is decided that the project needs to be ported to another platform and at the same time someone needs to start work on the beta for the next release.

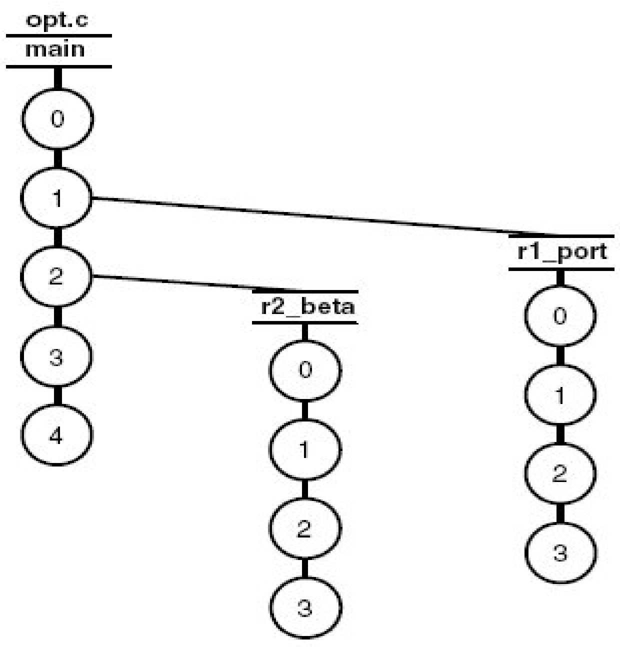


Figure 14: Branching

When using streams the branching of the elements is handled for you.

Sometime you may wish to use a private branch to perform a bug fix without affecting other developers perhaps.

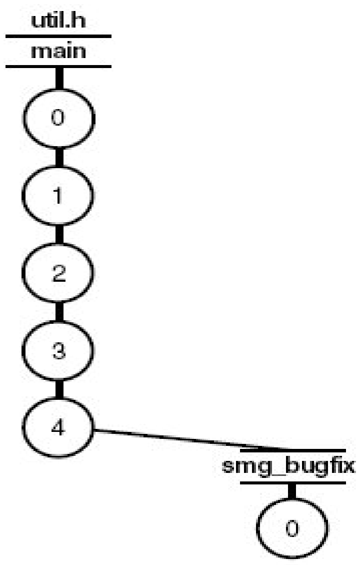


Figure 15: Private branch

If after some time of working a new branch you decide that it is no longer needed the branch can just be discarded and you can return to the original branch for your work. If you decide to keep the work then you will have to merge the branches back into the main branch. Remember to merge the branches in the order of dependence as shown in figure 13.

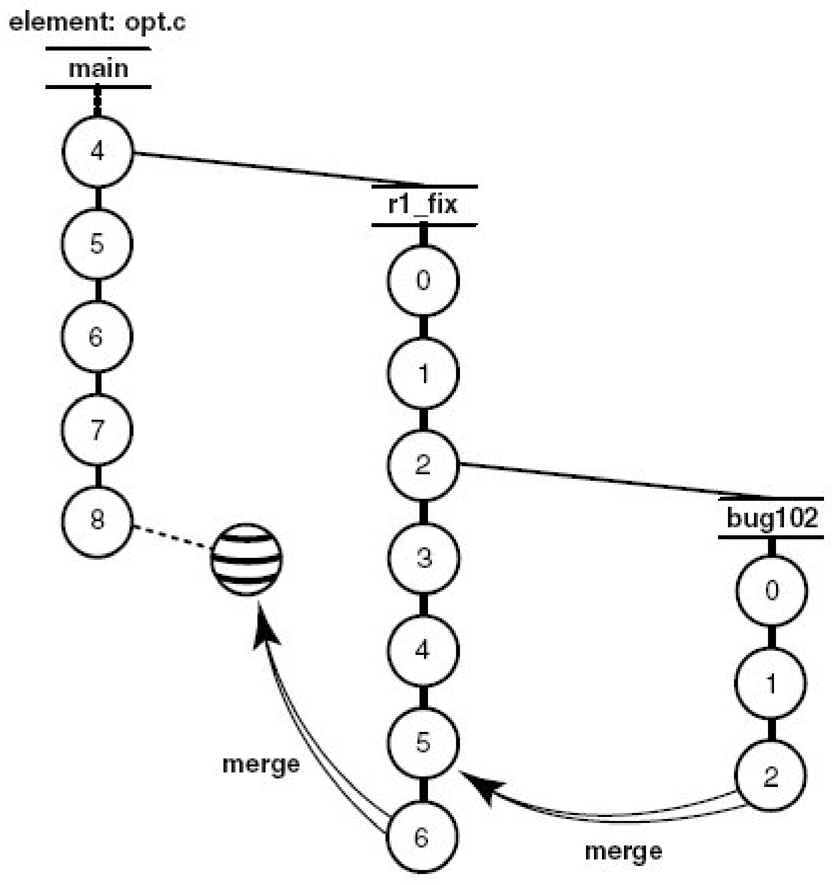
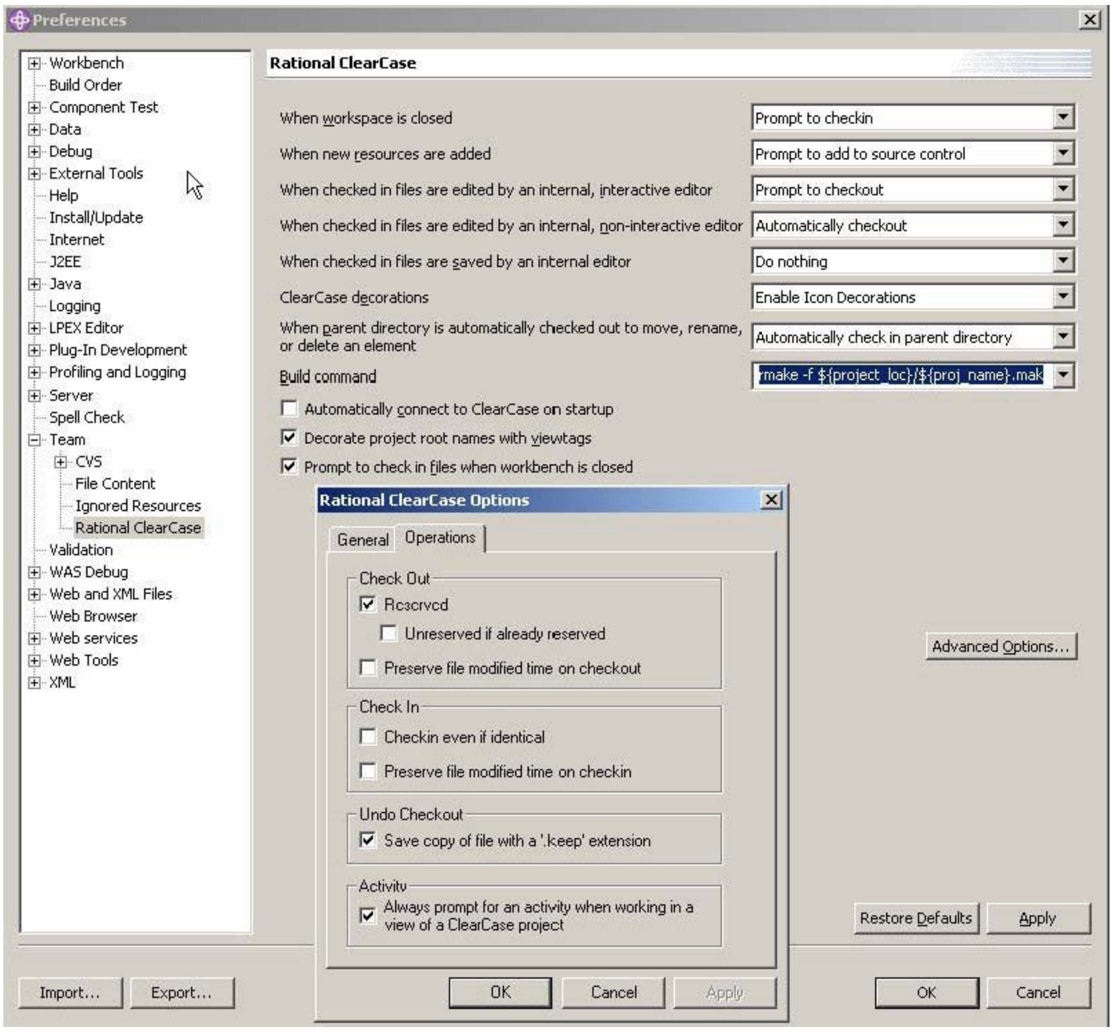


Figure 16 : Merging branches

4.4 ClearCase & WebSphere Business Integration Message Broker

WebSphere Business Integrator Message Broker uses the Message Broker Toolkit. The toolkit has the option of running with a number of SCM tools. Here we look at how we can use the toolkit as a ClearCase client.

Typically a VOB is created during the installation of ClearCase. It is usually the responsibility of the ClearCase administrator to create the necessary VOBs to connect to. You can however, set any preferences for ClearCase from inside your toolkit. To access these preferences select *Window -> Preferences* from the toolkit pull down menu. From there, click on *Team ->Rational ClearCase* to access the settings.



Rational ClearCase

When workspace is closed	Prompt to checkin
When new resources are added	Prompt to add to source control
When checked in files are edited by an internal, interactive editor	Prompt to checkout
When checked in files are edited by an internal, non-interactive editor	Automatically checkout
When checked in files are saved by an internal editor	Do nothing
ClearCase decorations	Enable Icon Decorations
When parent directory is automatically checked out to move, rename, or delete an element	Automatically check in parent directory
Build command	rmake -f ${project_loc}/${proj_name}.mak

☐ Automatically connect to ClearCase on startup
☑ Decorate project root names with viewtags
☑ Prompt to check in files when workbench is closed

Rational ClearCase Options — Operations

Check Out
☑ Reserved
☐ Unreserved if already reserved
☐ Preserve file modified time on checkout

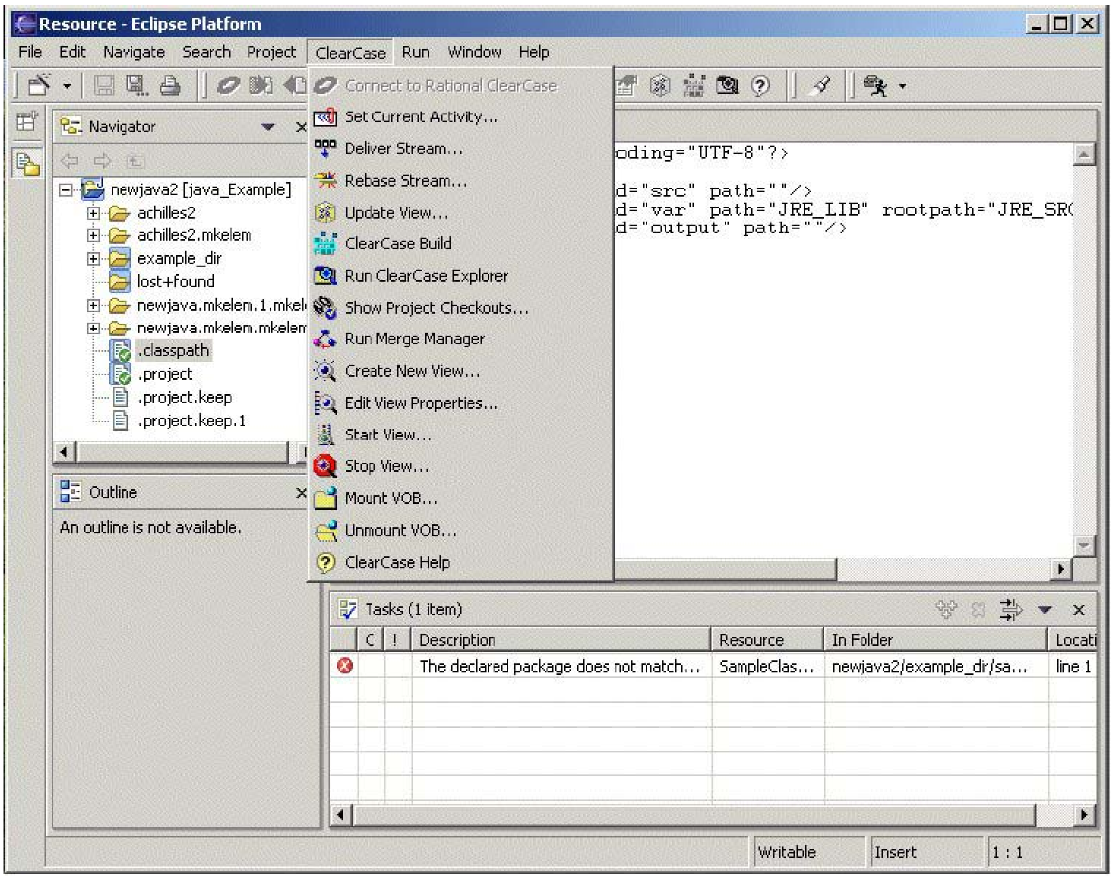
Check In
☐ Checkin even if identical
☐ Preserve file modified time on checkin

Undo Checkout
☑ Save copy of file with a '.keep' extension

Activity
☑ Always prompt for an activity when working in a view of a ClearCase project

Figure 17: ClearCase Settings

Rational ClearCase allows you to work in both on and offline modes. Before we can work in either we must first connect to ClearCase from inside the message broker toolkit. To do this we select the Connect to Rational ClearCase option from the ClearCase drop down menu inside the toolkit.



```
oding="UTF-8"?>

d="src" path=""/>
d="var" path="JRE_LIB" rootpath="JRE_SR(
d="output" path=""/>
```

	C	!	Description	Resource	In Folder	Locati
⊗			The declared package does not match...	SampleClas...	newjava2/example_dir/sa...	line 1

Figure 18 : ClearCase Integration

Once we are connected we can see all the options available to us. From inside the toolkit we have many functions available to us. Figure 16 shows the capabilities available from inside the toolkit.

Team Context Menu Options	*Rational ClearCase Menu Options*	*Rational ClearCase ToolBar Icons*
Add to Source Control	Connect to Rational ClearCase	Connect to Rational ClearCase
Check Out	Set Current Activity	Add to Source Control
Check In	Deliver Stream	Check Out
Undo Check Out	Rebase Stream	Check In
Update	Update View	Undo Check Out
Refresh Status	ClearCase Build *	Set Current Activity
Hijack	Run ClearCase Explorer	Deliver Stream
Compare With Previous Version	Show Project Checkouts	Rebase Stream
	Run Merge Manager	Refresh Status
Show Properties	Create New View	Update
Show Version Tree	Edit View Properties	Compare With Previous Version
Show History	Start/Stop View *	Show Version Tree
	Mount/Unmount VOB *	Show Properties
	ClearCase Help	Update View
		ClearCase Build *
		Run ClearCase Explorer
		ClearCase Help

Figure 19: CearCase capabilities

Some of the options are only available with the full ClearCase and they are marked *.

4.4.1 Creating a new project under ClearCase Control

To create a new version controlled project, create a project as normal via *File -> New -> Project* method. Remember to change the location of the project from the default directory to that of the VOB. The Add Elements to Source Control dialog box will now appear. Select the elements that need to be added.

4.4.2 Adding an existing project to ClearCase control

It may be that you have already projects started that you wish to be under Rational ClearCase control. To do this simply selecting the project and right clicking to bring up the team context menu then select *Team -> Share Project*. Choose Rational ClearCase and enter the path to the VOB.

4.4.3 Adding an existing ClearCase project to your workspace

There will be times that you will need to add projects from the VOB to your workspace. The usual process can achieve this. Select *File->Import* from the pull down menu in the toolkit. Then select *Existing Project into Workspace*. When the *Import Project* dialogue box appears specify the directory that is the root of the project in the VOB. The project should now appear in the toolkit workspace.

4.4.4 Adding an element to ClearCase control

From inside the toolkit it is possible to add new elements to ClearCase Control. This is done by right clicking on the element to get the Team context menu up. Choose *Team -> Add to Source Control*. This will bring up the *Add to Source Control* dialogue box where you place it under ClearCase control. If you are using UCM then it will also prompt for an activity.

4.4.5 Check out and edit

This can be done from either the Team context menu or the toolbar menu. Simply select the file you wish to edit and then checkout via either menu. The file can then be edited in the usual manor. If you are using UCM then you will be prompted to enter an activity as well.

It is also possible to have the files automatically check out when you edit them. This can be set up via the preference panels previously mentioned.

4.4.6 Save and Check in

Again this can be done from either the Team context menu or the toolbar menu. Once you have finished editing the file and saved it select the file. Then from either of the menus select check in. Don’t forget to add comments about changes that have been made.

4.5 Reference:

4.5.1 White Papers

Rational ClearCase Integration with WebSphere Studio TP731A

Integrating Rational ClearCase LT and WebSphere Studio Application Developer V5.

4.5.2 Redbooks

WebSphere Studio Application Developer Version 5 Programming Guide

Migration to WebSphere Business Integration Message Broker v5

4.5.3 Manuals

Rational ClearCase Rational ClearCase LT Introduction

Rational ClearCase Rational ClearCase LT Tutorial

4.5.4 Web-Based Courses

Getting Started with Rational ClearCase: Concepts and Terminology for WebSphere Studio v5 Users.

5 Working without a repository

This section provides suggested procedures for version and workspace management of WebSphere Integration Message Broker V5 objects. These procedures are aimed at a typical single user environment such as project creation for learning purposes, testing, demos and short-term development engagements where work is generally discarded or archived after an elapsed time period.

5.1 Organizing Projects

The Broker eclipse tooling keeps a default workspace for referencing all the projects that are created. Although the projects can be created and organized in separate folders, a single workspace poses a problem when it quickly gets cluttered with folders belonging to different projects becoming intermixed. As more and more projects are added, the list can become too large to handle, making it difficult to distinguish related projects.

The solution is to create a different workspace for each group of related projects such as client engagements, and trial message flows and sets; and to open the relevant workspace when launching the broker toolkit. This can be implemented easily by creating a different shortcut for each workspace.

The tooling creates the default workspace in:

```
      <Broker Install Directory>\eclipse\workspace\
```

The default workspace location for projects can be changed using the `-data` parameter in the shortcut as shown below:

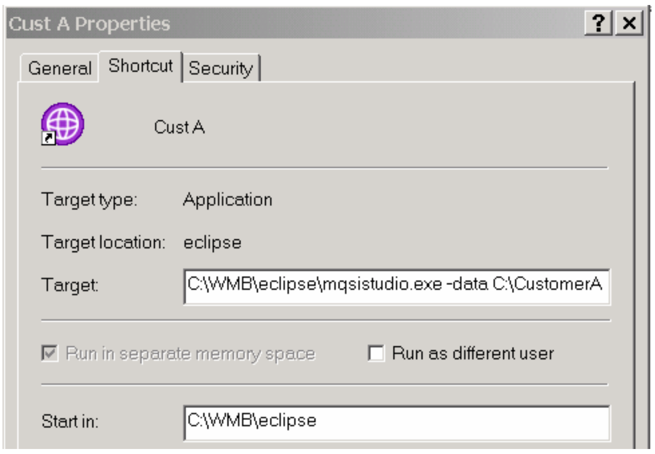
```
<Broker Install Directory>\eclipse\mqsistudio.exe -data <location>
```

The location can now be specified for example using a naming standard such as:

```
\<Customer A>\workspace

\<Customer B>\workspace

\<Demo>\workspace
```


Cust A Properties

General | Shortcut | Security

Cust A

Target type: Application

Target location: eclipse

Target: C:\WMB\eclipse\mqsistudio.exe -data C:\CustomerA

[✓] Run in separate memory space   [ ] Run as different user

Start in: C:\WMB\eclipse

The process can be repeated to create different shortcuts to launch the different workspaces:

On the first startup after setting the `-data` parameter, the broker will briefly display the message “Please wait…Completing the Install” to indicate that the new workspace is being initialized.

Tip: Other parameters, which can be used to customize the Message Brokers Toolkit Launcher, may be found by typing “org.eclipse.core.launcher.Main” in the help search panel.

5.2 Moving Projects into different workspaces

Once the different workspaces have been created, the projects, which previously belonged to one workspace, can be moved to their new workspaces. There are several techniques available.

1. Export project to a new workspace
2. Copy/Move project to a new workspace
3. Drag and Drop between Broker workspaces
4. Copy/Move using the Windows Explorer
5. Export/Import using ZIP files

5.2.1 Export Project using Toolkit

Using ‘Export’ the project can be copied directly to the new workspace location.

1. Select the project and choose “Export” from the File pull down menu.
2. Select “File System” from the “Export resources to the local file system” dialog panel and click Next
3. Specify the new workspace location in “Select Export destination” and select the “Create directory structure for files” from the options.
4. In the target workspace, create a new project and assign it the same name as before and the project should appear with its components in the new workspace.

To avoid ending up with the same project being referenced in two workspaces, delete the original project, making sure to select the “Also delete contents” check box.

5.2.2 Copy/Move Project using Broker Toolkit

This technique can be used to move projects between workspaces.

1. Select the project and right click to select move
2. Specify the new location belonging to the target workspace
3. When this has completed, the project is moved to the target directory but it doesn’t appear in the target workspace yet.
4. In the target workspace, create a new project and assign it the same name as before. An alternative method is to use `Import → Existing Project into workspace` and specify the directory in the file system.
5. The project should appear with its components in the new workspace.

To avoid ending up with the same project being referenced in two workspaces, delete the original project making sure to select the “Do not delete contents” check box, (applicable if project was moved).

5.2.3 Drag and Drop Projects between Broker Toolkits

Using this technique allows components to be selected and copied (or moved) to a different workspace. Unlike the other techniques, the selection is made on the project contents rather than the root project name.

1. In the target workspace, create a new project and assign a name to it.
2. Open the source project and make a selection on the required components
3. Use the drag and drop technique to copy or move the contents into the new workspace project.

5.2.4 Copy/Move Project using Windows Explorer

This technique can also be used for copying and moving projects between workspaces.

1. Copy or move the project root directory using the Windows file explorer from the original workspace location to the new location using drag and drop. (For Copy, press the CTRL key when dragging).
2. In the target workspace, create a new project and assign it the same name as before
3. The project should appear with its components in the new workspace.

5.2.5 Export Project as ZIP file

Exporting to a ZIP file enables several projects to be packaged and copied to a different workspace. As its name suggests, this method is more often used to copy projects to a different machine or archiving for backup purposes.

Using ‘Import’ to populate a project with the ZIP file contents creates a new folder within the project and may not be the desired result. In order to mirror the original folder structure, the ZIP file can be first unzipped to the new workspace directory outside the eclipse tooling and imported using eclipse `Import → Existing Project into workspace`.

5.3 Sharing projects between workspaces

You may need to access projects that you created in a different workspace. For example, you may have a message flow project called `AggregationExample` in a workspace in directory `C:\WBIMB Common code\workspace`.

To open the project from a different workspace, use `File → Import → Existing Project into workspace`. Navigate to the directory that contains the project, select it and import it. The project will appear in your workspace.

Note that `Import → Existing Project into workspace` does *not* copy the project into your workspace: it simply establishes a pointer from your workspace to the directory containing the project. **Any changes you make will modify the original project**. It is important that you recognize this distinction, as in all other forms of Import, the project you are importing is *copied* to your workspace; so you can think of the original as a "backup": if you make a mistake you can re-import. Again, this is not the case with `Existing Project into workspace`.