

Source Control with RTC and automation of build and deployment of WMB projects with Apache Ant

Table of Contents

Introduction.....	5
1 Background.....	6
1.1 Goal of SupportPac.....	6
1.2 Quick start.....	7
2 Introduction to Broker Archive file.....	8
2.1 Overview of the BAR file.....	8
2.2 Viewing BAR file from command line.....	8
2.3 Viewing BAR file in ToolKit.....	8
3 Building BAR file from command line.....	11
3.1 Description.....	11
3.2 Execution time.....	11
3.3 Execution workspace.....	11
4 Configurable properties in a BAR file.....	12
4.1 Viewing the configurable properties.....	12
4.1.1 WMB ToolKit view.....	12
4.1.2 Deployment descriptor document.....	13
5 Overriding configurable properties.....	14
5.1 GUI option.....	14
5.2 Command line option.....	14
5.3 Override properties file.....	14
5.4 Listing override properties of a BAR file.....	15
6 Deploying BAR file.....	16
6.1 Command line option.....	16
6.2 Command syntax.....	16
7 Sharing, checking-in and delivering changes in RTC.....	17
7.1 RTC Command line operations.....	17
Introduction to Ant and build process.....	18
8 Introduction to Ant.....	19
8.1 The build file.....	19
8.2 Sample build file.....	19
8.2.1 Invoking Ant.....	20
9 Deploying BAR file from Ant.....	22
9.1 Deployment on Windows and Unix platforms.....	22
9.2 Deployment on z/OS platform.....	22
9.2.1 Job scheduler – advantages and disadvantages.....	22
9.2.2 Stand-alone job – advantages and disadvantages.....	23
9.3 Different deployment paths.....	23
9.3.1 From Windows ToolKit.....	23
9.3.2 From Linux ToolKit.....	23
10 Ant tasks for different deployment paths.....	24
10.1 Windows BAT script.....	24
10.2 Linux shell script.....	24
10.3 Windows, Linux Java class.....	24
10.4 Windows, Linux triggering z/OS batch job.....	25
10.5 Windows, Linux submitting z/OS batch job.....	25
11 Adding external JARs and tasks for Ant.....	26
11.1 Add new Ant tasks.....	26
Using the Ant build files.....	28
12 WMB ToolKit on Windows, using BAT file for deployment.....	29
12.1 Build file and others.....	29
12.2 Description.....	29

Source Control with RTC and automation of build and deployment of WMB projects with Apache Ant

12.3 Using the build file.....	30
12.3.1 One-time set-up.....	30
12.3.2 Steps before initiating the build.....	30
12.3.3 Invoking Ant.....	31
12.3.4 Notes.....	31
13 WMB ToolKit on Linux, using shell script for deployment.....	32
13.1 Description.....	32
13.2 Using the build file.....	33
13.2.1 One-time set-up.....	33
13.2.2 Steps before initiating the build.....	33
13.2.3 Invoking Ant.....	34
13.2.4 Notes.....	34
14 WMB ToolKit on Windows, Linux using JAR file for deployment.....	35
14.1 Description.....	35
14.2 Using the build file.....	36
14.2.1 One-time set-up.....	36
14.2.2 Steps before initiating the build.....	36
14.2.3 Invoking Ant.....	37
14.2.4 Notes.....	37
15 WMB ToolKit on Windows, Linux triggering batch job for z/OS deployment.....	38
15.1 Description.....	38
15.2 Using the build file.....	39
15.2.1 One-time set-up.....	39
15.2.2 Steps before initiating the build.....	40
15.2.3 Invoking Ant.....	40
15.2.4 Notes.....	40
16 WMB ToolKit on Windows, Linux submitting batch job for z/OS deployment.....	41
16.1 Description.....	41
16.2 Using the build file.....	42
16.2.1 One-time set-up.....	42
16.2.2 Steps before initiating the build.....	43
16.2.3 Invoking Ant.....	44
16.2.4 Notes.....	44
17 Delivering change sets to RTC server.....	45
17.1 Description.....	45
17.2 Using the build file.....	45
17.2.1 One time set-up.....	46
17.2.2 Steps before initiating the build.....	46
17.2.3 Invoking Ant.....	46
17.2.4 Notes.....	46
Conclusion.....	47
18 Life-cycle of WMB project – deployment and source control.....	48
19 Bibliography.....	49
20 Author biography.....	49

Table of Contents - Figures

Fig 1 Paths of a WMB project.....	6
Fig 2: Contents of a Broker Archive (BAR) file.....	8
Fig 3 BAR file as seen in WMB ToolKit.....	8
Fig 4 BAR User log.....	9
Fig 5 BAR Service log.....	10
Fig 6: Configurable properties.....	12
Fig 7: Configurable properties as seen in broker.xml document.....	13
Fig 8: Sample output of mqsireadbar command.....	15
Fig 9: Ant with option for output file.....	21
Fig 10: Changing Ant preferences to add external JARs.....	26
Fig 11: Adding custom Ant task.....	27
Fig 12 Illustrative life-cycle of WMB project – deployment and source control.....	48

Table of Contents - Code

Code 1: Example of mqsicreatebar from Windows command line.....	11
Code 2: Sample syntax of mqsipplybaroverride command.....	14
Code 3: Sample Override properties file.....	14
Code 4: Sample syntax for mqsireadbar.....	15
Code 5: Sample syntax of mqsideploy command.....	16
Code 6: Sample Ant build file.....	20
Code 7: Ant task for deployment using Windows BAT file.....	24
Code 8: Ant task for deployment using Linux shell script.....	24
Code 9: Ant task for deployment using Java from Windows, Linux.....	24
Code 10: Ant task for triggering z/OS job from Windows, Linux.....	25
Code 11: Ant tasks for submitting batch job and retrieving status from Windows and Linux.....	25

Introduction

In this chapter, we produce the background for this SupportPac. We also introduce the following concepts that would be helpful in using the Ant build files. Advanced users of WebSphere Message Broker and Apache Ant may skip to [Using the Ant build files](#) chapter directly.

- [Broker Archive file](#)
- [Building Broker Archive file](#)
- [Configurable properties of a Broker Archive file](#)
- [Overriding configurable properties of a Broker Archive file](#)
- [Deploying Broker Archive file](#)
- [Sharing, checking-in and delivering changes to Rational Team Concert server](#)

1 Background

In this document, we will describe a procedure for automation of building, deploying and sharing WebSphere Message Broker (WMB) projects with Rational Team Concert (RTC) server. The automation will be done via the Apache Ant build tool embedded in both WMB Toolkit and RTC Eclipse client.

This SupportPac is applicable for WebSphere Message Broker version 7.0 and Rational Team Concert 3.0. Note that, WMB 7.0 Toolkit can be [integrated](#) with RTC 2.0 but not with RTC 3.0.

The Apache Ant build tool is an open source project for building projects and can be installed separately on one's work-station. However, in this SupportPac we are describing the Ant that is installed in the Toolkit and the Eclipse client. The version of Apache Ant does not have much of a bearing on this SupportPac because the Ant tasks used are simple ones.

1.1 Goal of SupportPac

A WMB project can follow two paths, viz. deployment to broker and sharing with a source control system. This is shown in the graphic below.

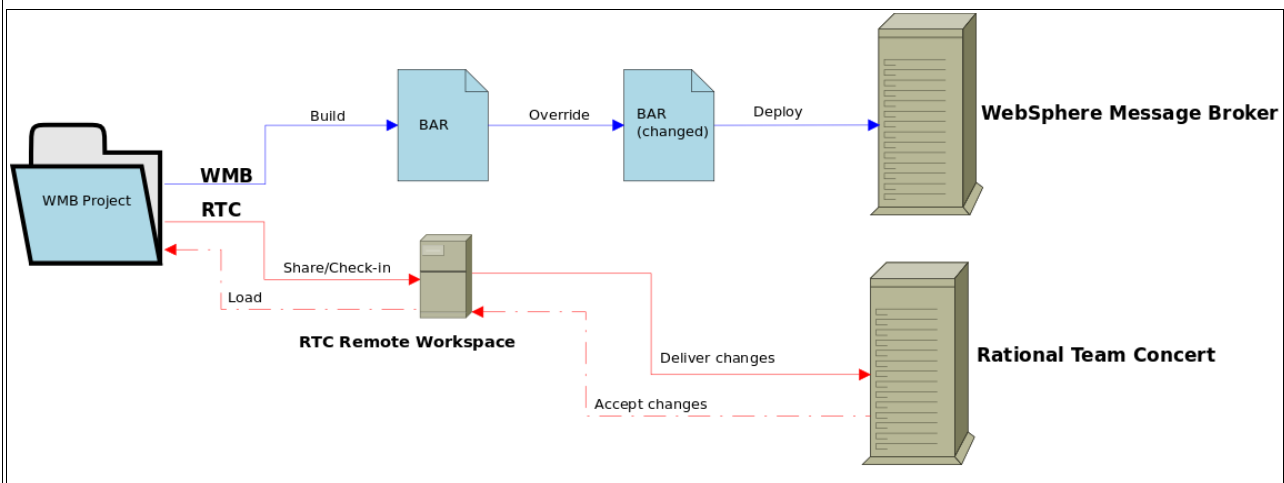


Fig 1 Paths of a WMB project

In this SupportPac, we present multiple Apache Ant build files that can be used for both of the paths (blue and red lines) in Fig 1 and for deployment scenarios (See Section 9.3). The path shown in blue line denotes the deployment to a WMB installation after the BAR file is built and its configurable properties overridden. The build files also cater to the option of only overriding the configurable properties and then deploying.

The path shown in red line denotes the sharing of projects to and from a RTC server. The sharing (or checking-in) is a two step process in general. First, all the changes for a given WMB project are shared or checked into a remote workspace. Then, after finalising all changes, these changes are associated with a work item and then deployed to the RTC server installation.

The path shown in red dashed lines is not automated.

1.2 Quick start

For advanced users of WMB and Apache Ant, to quickly getting started, use the build file and refer the section as mentioned below.

1. WMB ToolKit on Windows, using BAT file for deployment – `Sample-Windows-bat-build.xml`. See Section [12](#).
2. WMB ToolKit on Linux, using shell script for deployment – `Sample-Linux-sh-build.xml`. See Section [13](#).
3. WMB ToolKit on Windows, Linux using JAR file for deployment – `Sample-WMB-RTC-Java-build.xml`. See Section [14](#).
4. WMB ToolKit on Windows, Linux triggering batch job for z/OS deployment – `Sample-zOS-WMB-RTC-build.xml`. See Section [15](#).
5. WMB ToolKit on Windows, Linux submitting batch job for z/OS deployment – `Sample-CustomAnt-WMB-RTC-build.xml`. See Section [16](#).

2 Introduction to Broker Archive file

In this section, we discuss the Broker Archive (BAR) file.

2.1 Overview of the BAR file

The Broker Archive (BAR) file is a zipped file of various WMB artefacts. This zipped file is the only artefact that is deployed to the broker. Amongst the other possible contents of a BAR file are the message flows, message dictionaries and the Java `classpath` file.

Apart from these artefacts, the BAR file has a `META-INF` folder that contains the deployment descriptor document `broker.xml`. Other documents such as user logs, service logs etc. also form part of the BAR file.

2.2 Viewing BAR file from command line

```

nsubrahm@nsubrahm:~/Desktop$ unzip -l pubSubAuto.bar
Archive:  pubSubAuto.bar
  Length      Date    Time    Name
-----
         59   2011-07-04  11:12   META-INF/manifest.mf
       7067   2011-07-04  11:12   META-INF/broker.xml
       6840   2011-07-04  11:12   META-INF/service.log
       1262   2011-07-04  11:12   META-INF/user.log
       2338   2011-07-04  11:12   META-INF/bar-refresh.links
     1023976   2011-07-04  11:12   pubSubAuto.PubSubAutoMain.cmf
        2001   2011-07-04  11:12   Request-Payload.xsdzip
        4080   2011-07-04  11:12   PubSubAuto-00Java.jar
-----
     1047623                   8 files
  
```

Fig 2: Contents of a Broker Archive (BAR) file

Here is a screen-shot of a BAR file with its contents listed out from a Linux command prompt. Note the `.jar` and the `.xsdzip` files for Java component and message sets respectively. The message flow will appear as `.cmf` (compiled message flow) file. To view the contents of the BAR file on a Windows machine, change the file extension to `.zip` and then open it with 'Compressed (zipped) Folder', WinZip, 7zip, etc.

2.3 Viewing BAR file in ToolKit

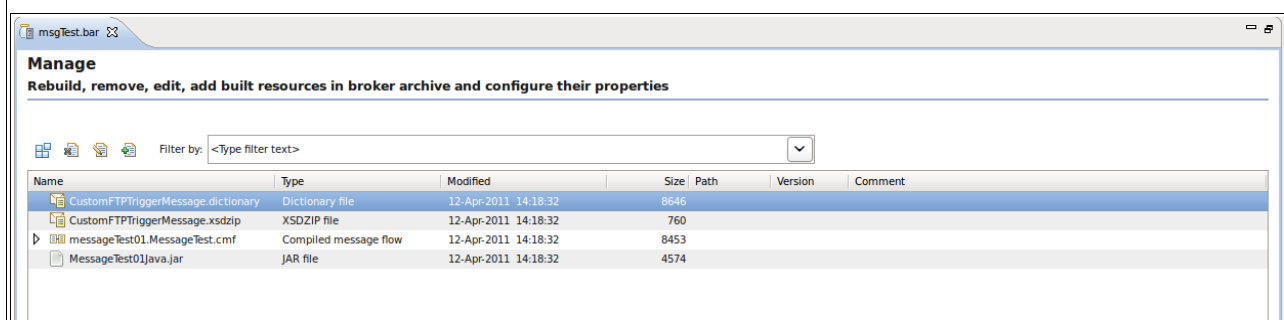


Fig 3 BAR file as seen in WMB ToolKit.

The contents of the BAR file can also be seen in the WMB ToolKit. The screen-shot below shows the user log.

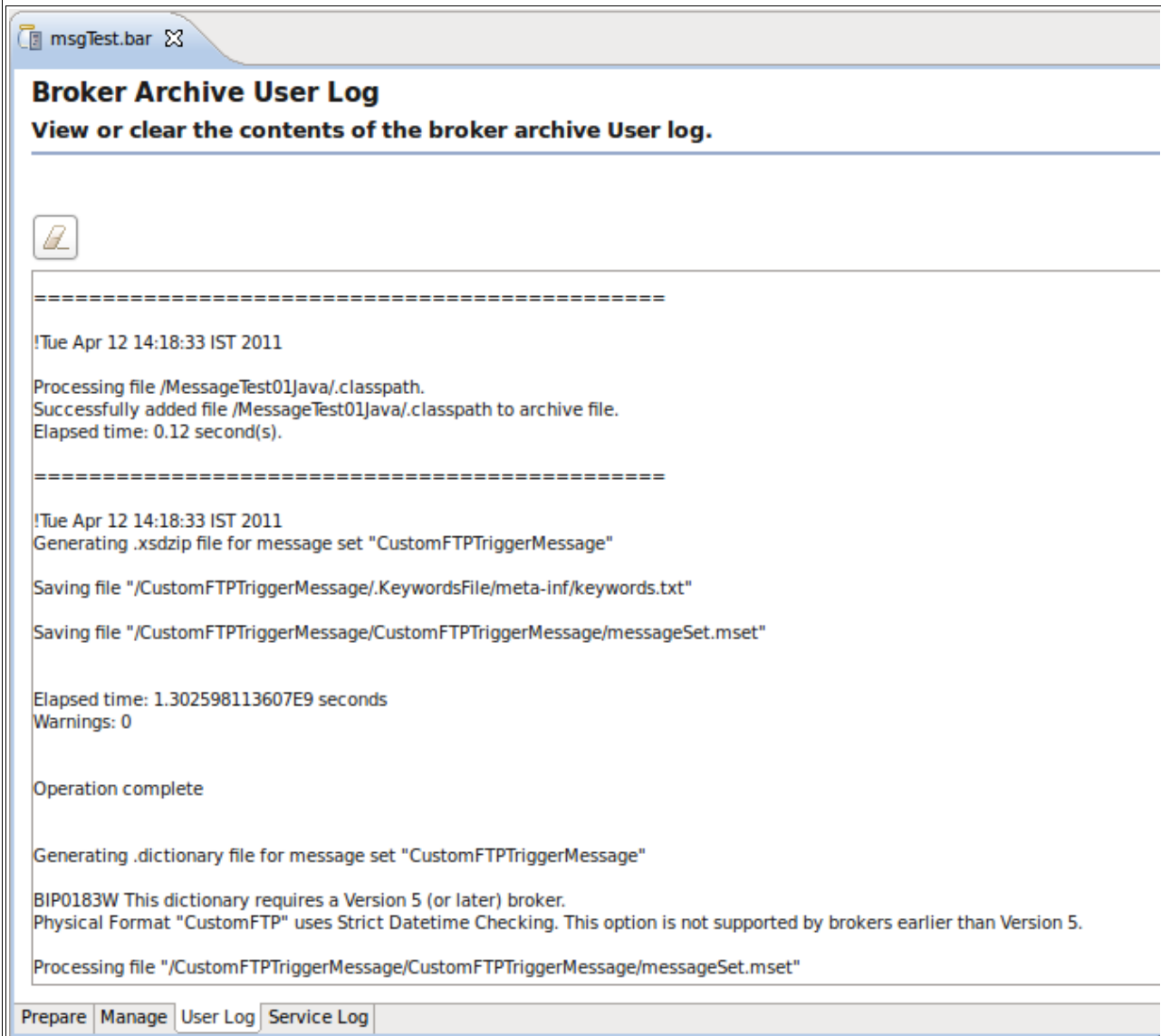


Fig 4 BAR User log

Note the log entry above for `.classpath` file. When a WMB project has a corresponding Java project, then the JAR file is built using the `.classpath` file and then added to the BAR file. The screen-shot below shows the service log.

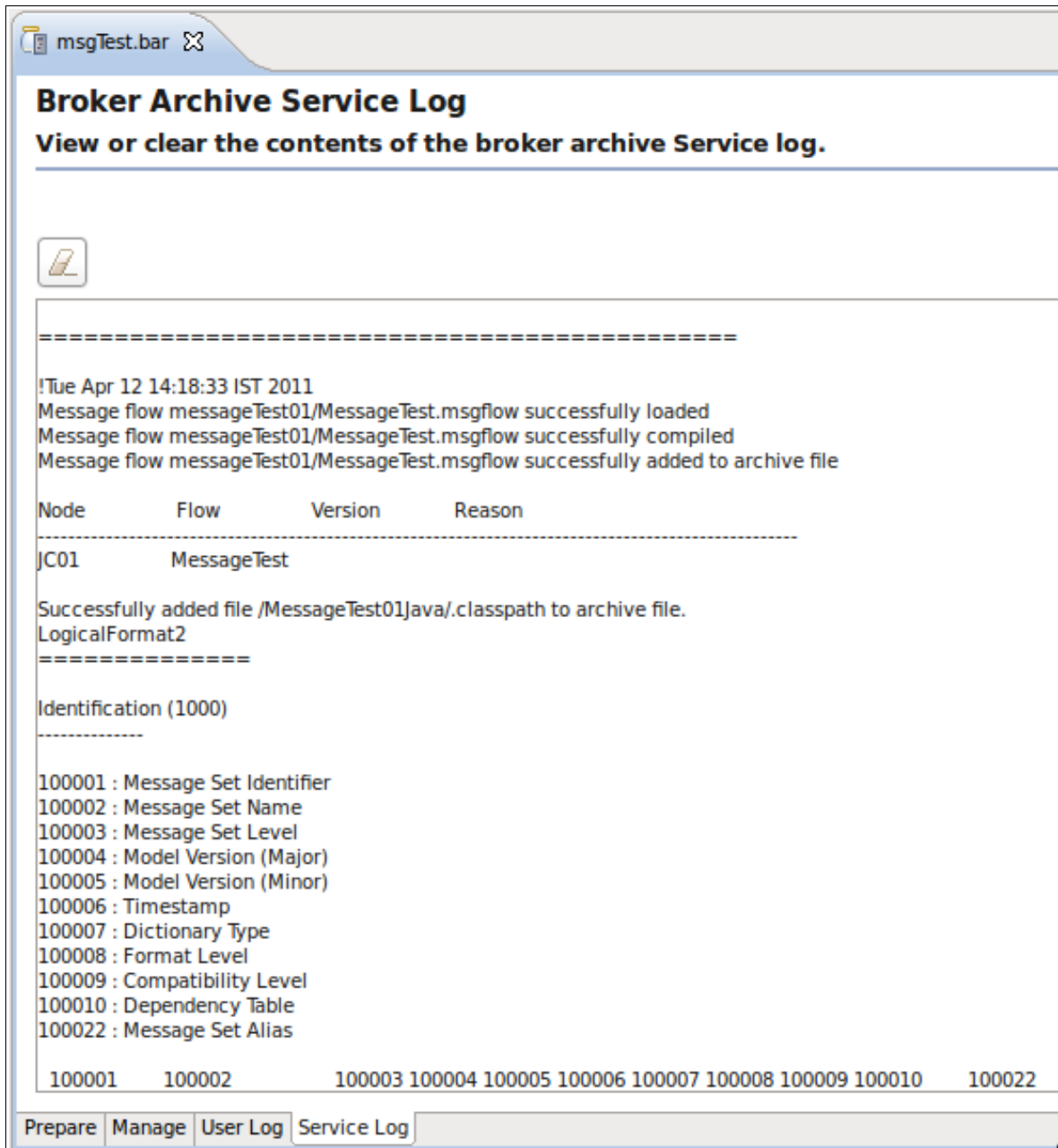


Fig 5 BAR Service log

Note the log entries for .classpath file and .xsdzip file. The former is for associated Java project and the latter for the multiple XSD documents used for the message set.

3 Building BAR file from command line

3.1 Description

The BAR file can be built from the command line of the local machine using the WMB ToolKit command `mqsicreatebar`. It would normally be available at `C:\Program Files\IBM\WMBT700` and is documented [here](#). To understand this command better, we take a simple example.

```
mqsicreatebar -data C:\workspace-wmb -b C:\BAR\fileReader.bar -cleanBuild -p  
FileReader -o FileReader\fileReader\fileReader.msgflow
```

Code 1: Example of mqsicreatebar from Windows command line

In the command Code [1](#), we pass the name of the work space, the absolute path of the BAR file to be created, the project name followed by actual file name. These file names are passed relative to the work space location.

The name of the workspace used should be different from the one being used in the ToolKit. This is because, the `mqsicreatebar` command actually runs the Eclipse client itself in a 'headless' mode. Therefore, the same workspace gets used by two Eclipse sessions causing errors in build process.

Multiple project names and file names can be provided as a continuous list. An example could be a project that uses a message set project and perhaps a Java project too. In this case, the project names for both the message set and Java projects need to be passed. The file names to be passed are the `messageSet.mset` file and `.classpath` for the message set and Java projects respectively.

3.2 Execution time

It will be noted that when the BAR file is built from the command line, the time taken is often noticeably longer than time taken when built from the WMB ToolKit. This is because the `mqsicreatebar` command actually runs the Eclipse client itself in a 'headless' mode. The Eclipse front end probably keeps track of various file names, project references and lots of other parameters that speeds up the build time. As this does not happen from command line, the time taken is noticeably longer.

3.3 Execution workspace

The `mqsicreatebar` command uses a directory location as an Eclipse workspace. For successful execution, it is [highly recommended](#) that, the workspace should always be created afresh regardless of completion of build process. Thus, whether or not the BAR file gets created, it is a good practice to specifically delete the workspace after running `mqsicreatebar` command. Consequentially, the WMB projects should be exported to this workspace before building the BAR.

4 Configurable properties in a BAR file

Every node in a message flow has certain [configurable properties](#) that can be altered once a BAR file has been created. The configurable properties provide a facility to change certain values of nodes without having to re-build the whole WMB project. For example, in a `MQOutput` node, the name of the queue manager and the queue name are configurable properties. These values can be changed before deploying the BAR file to say, a system testing region or a production region because the names of the queue manager and the queue name on these regions could be different. Another example could be name of the data source used in database related nodes. This name could be changed when deploying the BAR file to different regions.

Similarly, the values of any user-defined properties could also be changed.

4.1 Viewing the configurable properties

The configurable properties can be seen in the BAR file in the WMB ToolKit. Or, one can inspect the deployment descriptor document (`broker.xml`) in the BAR file.

4.1.1 WMB ToolKit view

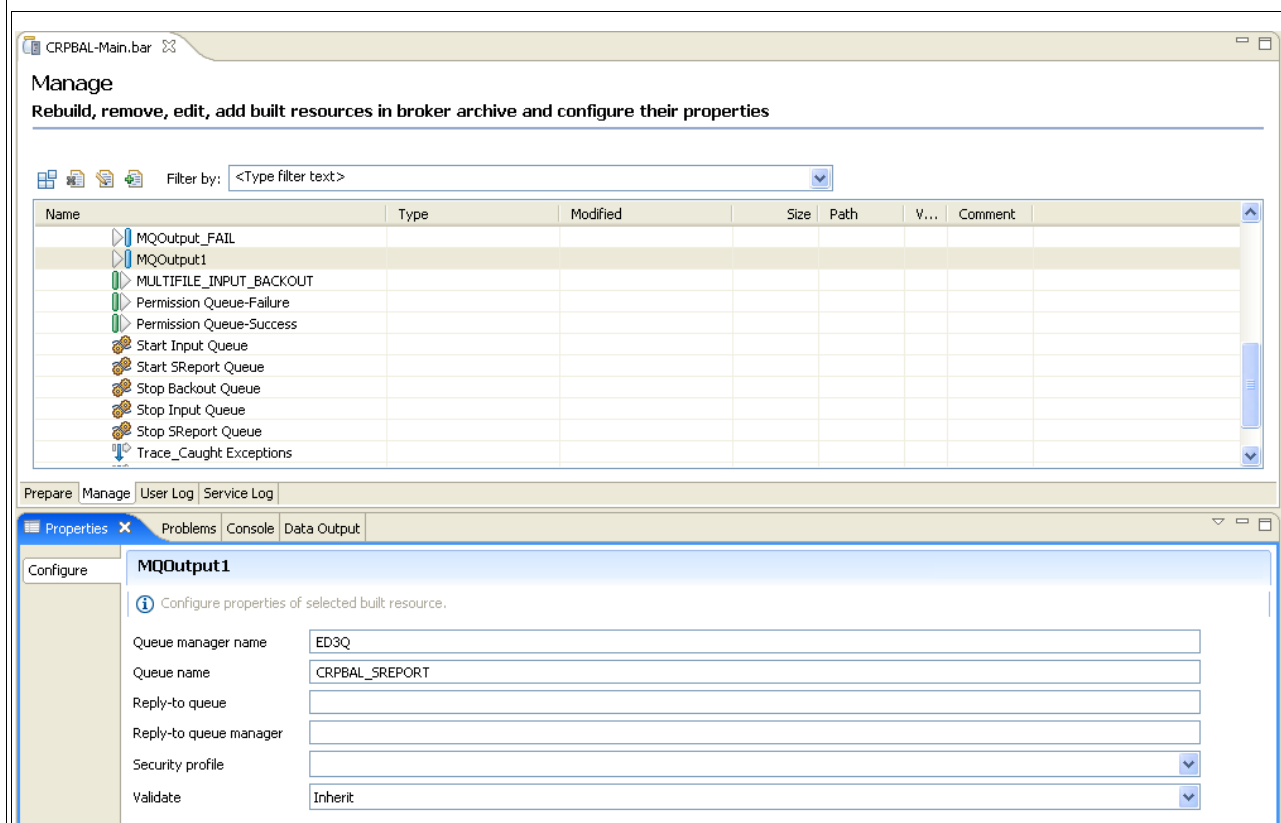


Fig 6: Configurable properties

In the screen-shot above, we see the configurable properties of an `MQOutput` node named as `MQOutput1`. These nodes themselves comprise a compiled message flow (`.cmf`) file.

4.1.2 Deployment descriptor document

Every BAR file has a deployment descriptor document named as `broker.xml`. Amongst others, this document contains information about all the configurable properties. In Fig 6, two properties have been marked in red boxes. The first one is the queue name as seen in the Fig 5. The second one is for reply-to queue manager as also seen in Fig 5 and is unpopulated. Consequently, the first property appears with the attribute `override` and the second appears with `uri`. Both these properties can be altered. Also, the override can be applied for a value that is to be changed throughout the deployment descriptor document regardless of nodes. Or, it can be changed for a specific node.

```

<ConfigurableProperty override="/EUB1/WIM/CRP/WBAL/BACKUP/" uri="CRPBAL_MAIN_12_B1#arg target"/>
<ConfigurableProperty override="CRPBAL_SREPORT" uri="CRPBAL_MAIN_12_B1#MQOutput1.queueName"/>
<ConfigurableProperty uri="CRPBAL_MAIN_12_B1#EmailOutput1.securityIdentity"/>
<ConfigurableProperty uri="CRPBAL_MAIN_12_B1#Start SReport Queue.validateMaster"/>
<ConfigurableProperty override="CRPBAL_SPERMISSION" uri="CRPBAL_MAIN_12_B1#Permission Queue-Success.queueName"/>
<ConfigurableProperty override="/EUB1/WIM/CRP/WBAL/OUTPUT/*" uri="CRPBAL_MAIN_12_B1#arg.source"/>
<ConfigurableProperty uri="CRPBAL_MAIN_12_B1#MQOutput_BACKOUT.validateMaster"/>
<ConfigurableProperty uri="CRPBAL_MAIN_12_B1#Permission Queue-Success.topicProperty"/>
<ConfigurableProperty uri="CRPBAL_MAIN_12_B1#Permission Queue-Failure.serializationToken"/>
<ConfigurableProperty override="/EUB1/WIM/CRP/WBAL/ExceptionList.log" uri="CRPBAL_MAIN_12_B1#Trace_Caught Exceptions.filePath"/>
<ConfigurableProperty uri="CRPBAL_MAIN_12_B1#MULTIFILE_INPUT_BACKOUT.serializationToken"/>
<ConfigurableProperty uri="CRPBAL_MAIN_12_B1#MQOutput_CATCH.replyToQMgr"/>
<ConfigurableProperty uri="CRPBAL_MAIN_12_B1#Stop SReport Queue.validateMaster"/>
<ConfigurableProperty uri="CRPBAL_MAIN_12_B1#MQOutput1.replyToQMgr"/>

```

Fig 7: Configurable properties as seen in `broker.xml` document

5 Overriding configurable properties

5.1 GUI option

The simplest way to apply overriding values of the configurable properties in a BAR file is to edit the same in WMB ToolKit. See [4.1.1](#).

5.2 Command line option

The override can also be applied by executing the command `mqsiapplybaroverride` from command line. This command is also a ToolKit command and hence, can be generally found at `C:\Program Files\IBM\WMBT700`. This command is documented [here](#). To understand this command better, we take a simple example.

```
mqsiapplybaroverride -b C:\BAR\fileReader.bar -p C:\props\BAR-override.props
```

Code 2: Sample syntax of mqsiapplybaroverride command.

In [Code 2](#), we specify the name of the BAR file in which the configurable properties have to be changed and also a file name that has pairs of name and value whose properties have to be changed.

As documented for the command syntax, the name value pairs can also be provided as a list on the command line also or as a deployment descriptor document or even a BAR file also.

5.3 Override properties file

In this file, we provide a name-value pair for the properties whose values would have to be changed for deployment. A sample properties file has been shown below.

```
EB1Q=ED3Q
pubSubAuto.PubSubAutoRegMail#DI01.dataSource=UKPROD
```

Code 3: Sample Override properties file.

Note that, the overrides are of two kinds. One, changing the overridden value throughout the deployment descriptor and two, changing a specific property of a given node in the message flow. Also, the values of user-defined properties could be changed accordingly.

These name value pairs could be the name of the override and its corresponding values or the URI and its corresponding values. With the former, one can change all the nodes that have a given override value with a new value. For example, by providing a new value as `PRODDB` to an existing override value as `DEVDB` will change all occurrences of `DEVDB` for all nodes and properties. Therefore, with this option, one can make a comprehensive change to say database name from development to production of all nodes in a message flow, etc.

With the latter option, the change can be limited to a specific property of a specific node. For example, the reply-to queue manager of one particular `MQInput` node can be changed to accommodate system testing, etc.

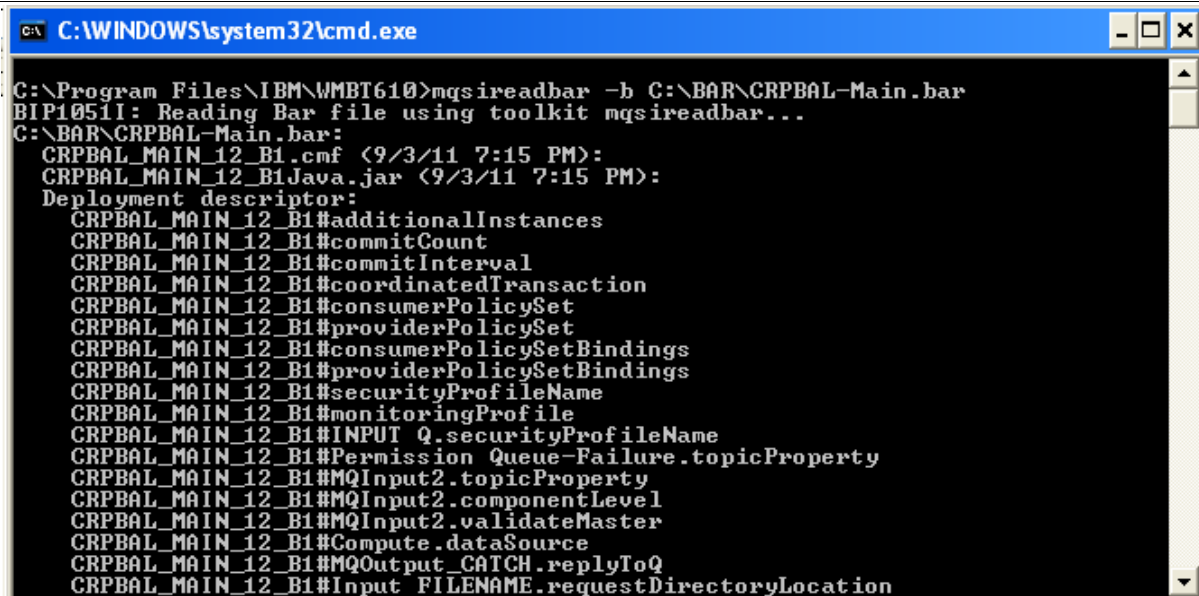
The URI syntax to be used is same as seen in deployment descriptor; that is, the broker schema followed by a period followed by the message flow name followed by the hash (#) symbol, the node name, a period and then the actual the property to be overridden.

5.4 Listing override properties of a BAR file

Using `mqsireadbar` command, one has an option to list out the configurable properties. This is also a Toolkit command and is therefore available at `C:\Program Files\IBM\WMBT700`. This list can be handy in finding the exact name of the property to be overridden. Or, to see the override values currently in effect. A sample [command syntax](#) has been shown below.

```
mqsireadbar.exe -b C:\BAR\CRPBAL-Main.bar
```

Code 4: Sample syntax for `mqsireadbar`



```
C:\WINDOWS\system32\cmd.exe
C:\Program Files\IBM\WMBT610>mqsireadbar -b C:\BAR\CRPBAL-Main.bar
BIP1051I: Reading Bar file using toolkit mqsireadbar...
C:\BAR\CRPBAL-Main.bar:
CRPBAL_MAIN_12_B1.cmf <9/3/11 7:15 PM>:
CRPBAL_MAIN_12_B1Java.jar <9/3/11 7:15 PM>:
Deployment descriptor:
CRPBAL_MAIN_12_B1#additionalInstances
CRPBAL_MAIN_12_B1#commitCount
CRPBAL_MAIN_12_B1#commitInterval
CRPBAL_MAIN_12_B1#coordinatedTransaction
CRPBAL_MAIN_12_B1#consumerPolicySet
CRPBAL_MAIN_12_B1#providerPolicySet
CRPBAL_MAIN_12_B1#consumerPolicySetBindings
CRPBAL_MAIN_12_B1#providerPolicySetBindings
CRPBAL_MAIN_12_B1#securityProfileName
CRPBAL_MAIN_12_B1#monitoringProfile
CRPBAL_MAIN_12_B1#INPUT Q.securityProfileName
CRPBAL_MAIN_12_B1#Permission Queue-Failure.topicProperty
CRPBAL_MAIN_12_B1#MQInput2.topicProperty
CRPBAL_MAIN_12_B1#MQInput2.componentLevel
CRPBAL_MAIN_12_B1#MQInput2.validateMaster
CRPBAL_MAIN_12_B1#Compute.dataSource
CRPBAL_MAIN_12_B1#MQOutput_CATCH.replyToQ
CRPBAL_MAIN_12_B1#Input FILENAME.requestDirectoryLocation
```

Fig 8: Sample output of `mqsireadbar` command

For the sake of brevity, only a portion of the output of the above command has been shown.

6 Deploying BAR file

The simplest way to deploy a BAR file is from the WMB ToolKit.

6.1 Command line option

A BAR file can be deployed from the command line using the `mqsdeploy` command. This is a MQSI command and thus, has two important factors to be taken care of. Firstly, this command is available only when a local broker is also installed. Therefore, the path for this command is `C:\Program Files\IBM\MQSI\7.0\bin` which is different from the WMB ToolKit commands' location.

Secondly, to run `mqsdeploy`, a proper command environment must exist. See [here](#). To set up the command environment in Windows, first the `mqsiprofile` command would have to be run. And, then, in the same session, the `mqsdeploy` command should be run. The `mqsiprofile` command is also a MQSI command and hence it is also available in the MQSI installation folder.

The `mqsdeploy` command is documented [here](#).

6.2 Command syntax

Amongst the various syntax options available for `mqsdeploy`, we would select the one for broker deployment from the local workstation. To do so, we need to pass the IP address, the port number, the name queue manager that is listening at this IP address and port number. The name of the broker hosted here is also mentioned. The BAR file name that is to be deployed and the execution group to which it would be deployed is also passed.

```
mqsdeploy -i ip.address.of.broker -p 1414 -q QMGR -e EXECGRP -a  
C:\BAR\BARFile.bar
```

Code 5: Sample syntax of `mqsdeploy` command

7 Sharing, checking-in and delivering changes in RTC

A WMB project can be shared in RTC if it was not ever in RTC earlier. New changes to the project may be checked in. While sharing or checking-in the projects with the repository workspace in RTC, the changes would be collected into a change set. The changes flow from the repository workspace to the RTC stream only when they are specifically delivered. To deliver the changes, we assume a policy of associating every change with a RTC work item.

7.1 RTC Command line operations

The RTC Eclipse client provides a rich GUI for all the operations of sharing, checking-in and delivering changes. However, as we are discussing an automated solution using Apache Ant, we would opt for the command line option. The command to be run is `scm` which is generally available at `C:\Program Files\IBM\TeamConcert\scmtools\eclipse`. Some examples of running the `scm` command is available [here](#).

Introduction to Ant and build process

In this chapter, we introduce Apache Ant – a tool for automation of build process. We also discuss in detail the Ant build process as required for WebSphere Message Broker projects. The following topics are covered.

- Introduction to Ant
- Ant tasks for different deployment paths
- Adding external JARs and tasks for Ant

8 Introduction to Ant

[Apache Ant](#) is a Java based open source software build tool. It was originally designed for building Java applications. But, it can be used for building any application. In its goal of building applications, it is similar to `make` command in Unix machines. As Ant is based on Java and uses XML for the build file, it becomes a platform independent build tool.

The Ant build tool comes pre-packaged with the WMB ToolKit and RTC Eclipse client and can be accessed in an appropriate view of the Eclipse client. Although, Ant can be invoked from command line also.

The Ant build tool is thoroughly documented [here](#) and the manual is [available for free download](#) as an archive of HTML files in various zipped formats.

8.1 The build file

The Ant build tool uses an XML document for input to the build process. This build file is generally named as `build.xml` although, one can use any name. Each build file contains one *project* and at least one *target*. Each such target can have one or more *tasks*.

An Ant task is an executable piece of code that is invoked by Ant. There is a good number of built-in tasks available in Ant. Examples of such tasks are copying, moving, deleting files and folders. Other examples include FTP, invoking a Java class, a shell command, Windows BAT file or just about any executable on a given platform. These tasks are run sequentially in a given target.

An Ant target in the build file comprises one or more tasks and defines one particular step in the overall build process. For example, if you are building an executable from C source files, you would first compile source files individually. Then, link all the compiled output together to create the executable. You might then set file permissions for the executable and finally check-in into a source control system. It is obvious that, there is a dependency between these steps. That is, you cannot link the compiled output until the compilation is successfully over. Similarly, in Ant too, each target can have dependencies defined to them so that the order of execution is clear.

An Ant project has at least one (default) target that is always executed when Ant is invoked without specifying any target name.

An Ant build file also has a set of properties which is a pair of name and value. These properties provide a very easy way to customize the build file. The properties can be set in the project before any targets as this allows the propagation of properties. They can also be set in a particular target. The properties can be defined by simply coding in the build file directly or loading from an external file, etc.

8.2 Sample build file

Here is a sample build file copied from the Ant manual. It shows the usage of targets, their dependencies and tasks. Some points of note have been marked in blue.

Source Control with RTC and automation of build and deployment of WMB projects with Apache Ant

```
<project name="SampleProject" default="dist" basedir=". ">
  <description>simple example build file</description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist" location="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init" description="compile the source " >
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile" description="generate the distribution" >
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>


    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>


  <target name="clean" description="clean up" >
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

Code 6: Sample Ant build file

8.2.1 Invoking Ant

The Apache Ant build tool can be invoked from the command line with the build file as one of the input arguments. Alternatively, the WMB ToolKit and RTC Eclipse client have an Ant view, where one can add the build file and execute them. By default, the messages from execution are displayed on the command console (when invoking from command line) or to the console in the Eclipse perspective (when invoked from the ToolKit). This behaviour can be easily changed such that the messages are directed to a file. This can be desirable if a log of a build process should be maintained.

In the WMB ToolKit, first, add the build file in the Ant. To add the build file, first the Ant view should be opened. To do this, go to **Window** → **Show view ...** → **Other ...** . Expand **Ant** in the resulting dialogues box, select **Ant** and click **OK**. In the Ant view, click on  to add the build file.

Then, to invoke Ant, simply click on  icon. If no targets were selected, then the default target would be run. If a target were selected, then it would be run *after* meeting all dependencies, if any.

To see the results of invocation, look at the Console view. If this view does not exist in the perspective already, it can be added as follows. Go to **Window** → **Show view ...** → **Other ...** . Expand **General** in the resulting dialogue box, select **Console** and click **OK**.

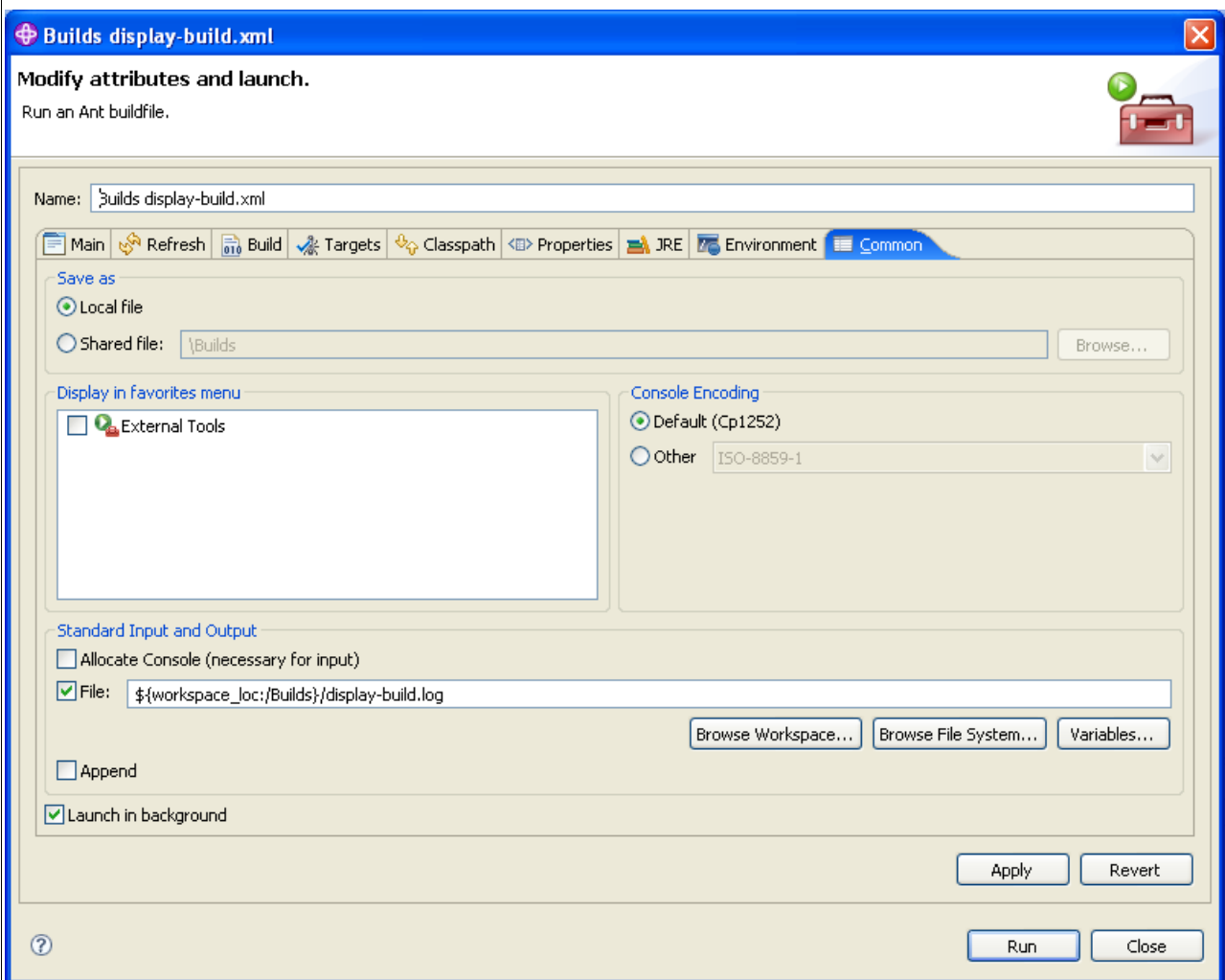


Fig 9: Ant with option for output file

9 Deploying BAR file from Ant

As discussed in Section [6.1](#), a proper command environment needs to be set-up before invoking `mqsidedeploy` to deploy the BAR file. This brings up a challenge when deploying from Ant. Both the commands for setting up the profile and deploying can be invoked as `exec` tasks from Ant. However, the moment the task for setting the profile is over, the environment is not cascaded over to the next (deployment) command. Therefore, all the environment variables that are set, after the profile command is run, no longer holds for the deployment command. In other words, there is no option to cascade the environment from one Ant task to another.

9.1 Deployment on Windows and Unix platforms

To get around this limitation, on Windows platform, a Windows BAT file that runs both the commands for setting up the profile and deploying is coded. This BAT file is then invoked from the Ant `exec` task with arguments passed for the names of execution groups and the name of the BAR file.

On a Linux/Unix platform, the commands for setting up the profile and deploying should be encapsulated in a shell script.

One can also develop Java classes with Configuration Manager Proxy (CMP) API to do the deployment. The Java option makes the deployment procedure platform independent in so far as the execution environment is considered. Once the Java application using CMP API is prepared, it can be invoked from Ant using the `java` task.

9.2 Deployment on z/OS platform

For WMB running on z/OS platform, one can opt for submitting batch jobs for deployment. For WMB 7.0, the list of jobs are documented [here](#).

When deploying from z/OS, a decision should be taken early on as to whether the jobs are to be compulsorily submitted using a job scheduler product. Or, whether the jobs can be submitted from outside of a job scheduler. Thus, the choices are:

1. Create a file with parameters for overriding the configurable properties and deployment. Send this file over to z/OS using FTP to a z/OS dataset for which a dataset trigger exists in the job scheduler. The job scheduler submits a job that uses the file sent via FTP and has steps for overriding the configurable properties and deployment.
2. Create a file with required job steps on the local workstation. Using this file as a template, create a new file by appropriately replacing text for overriding the configurable properties and deployment. Send this new file via FTP as a job.

The second option exploits the ability to submit a batch job from local workstation using FTP. See [here](#) for a sample implementation. However, the `ftp` Ant task does not have the feature of passing the `quote` command that enables job submission. Therefore, the second option cannot be used in Ant without creating a custom task.

As both of the options above are dependent on FTP, there is a dependency on external libraries that provide support for FTP. The libraries used are from [Apache Commons-Net](#).

9.2.1 Job scheduler – advantages and disadvantages

The option of using a job scheduler makes adherence to programming standards easy. Another advantage of using a job scheduler is that, a single copy or version of the job is available for all

developers. Any changes in future, therefore, needs to be done only at one place.

The job to be submitted is governed by a dataset trigger. That is, the job for overriding the configurable properties and deployment can begin only when a given dataset is created. If there are multiple users trying to submit jobs at the same time, then contentions may occur.

9.2.2 Stand-alone job – advantages and disadvantages

With a stand-alone job, there will never be an issue of contentions as discussed in [11.2.1](#) above as every user will submit his/her own copy of the job.

While not a clear disadvantage, but at some installations, the requirement of having a copy of the job on local workstation may not be well-received. This option also has the overhead of maintaining the custom task for submitting job from FTP.

9.3 Different deployment paths

The following deployment paths exist.

9.3.1 From Windows ToolKit

1. Using a BAT file, the deployment is done with the commands encapsulated in a Windows BAT file. The BAT file is invoked by the `exec` task in an Ant build file.
2. Using a Java JAR file, the deployment is done through a JAR file that uses CMP API. The Java class is invoked by the `java` task in an Ant build file.
3. The deployment is done via a batch job triggered by the creation of a dataset that holds parameters for deployment. This dataset is created by `ftp` task in an Ant build file.
4. The deployment is done by submitting the batch job and retrieving its status using a custom Ant task.

9.3.2 From Linux ToolKit

1. Using a shell script, the deployment is done with the commands encapsulated in a Unix shell script. The shell script is invoked by the `exec` task in an Ant build file.
2. Using a Java JAR file, the deployment is done through a JAR file that uses CMP API. The Java class is invoked by the `java` task in an Ant build file.
3. The deployment is done via a batch job triggered by the creation of a dataset that holds parameters for deployment. This dataset is created by `ftp` task in an Ant build file.
4. The deployment is done by submitting the batch job and retrieving its status using a custom Ant task.

10 Ant tasks for different deployment paths

Based on the deployment paths discussed in [9.3](#), the Ant task for deployment could be different.

10.1 Windows BAT script

```
<target name="deployBAR" description="Deploying BAR file">
  <echo message="Deploying Broker Archive file : ${bar.filepath}" />
  <exec executable="cmd failonerror="true">
    <arg line="/c deployBAR.BAT ${exec.group} ${bar.filepath}" />
  </exec>
</target>
```

Code 7: Ant task for deployment using Windows BAT file

The name of the executable is `cmd` which invokes a BAT file named `deployBAR.BAT` with the name of the execution group and BAR file name as arguments. These arguments have been shown as Ant properties names.

10.2 Linux shell script

```
<target name="deployBAR" description="Deploying BAR file">
  <echo message="Deploying Broker Archive file : ${bar.filepath}" />
  <exec executable="/bin/bash" failonerror="true">
    <arg line="deployBAR.sh ${exec.group} ${bar.filepath}" />
  </exec>
</target>
```

Code 8: Ant task for deployment using Linux shell script

The name of the executable is `/bin/bash` which invokes a shell script named `deployBAR.sh` with the name of the execution group and BAR file name as arguments. These arguments have been shown as Ant properties names.

10.3 Windows, Linux Java class

```
<java classname="${class.name}" fork="true">
  <classpath>
    <pathelement location="${wmb.jars}\ConfigManagerProxy.jar"/>
    <pathelement location="${wmg.jars}\com.ibm.mq.jar"/>
    <pathelement location="${deploy.jar}\deployBAR-${wmb.version}.jar"/>
  </classpath>
  <arg path="${bar.filepath}"/>
  <arg value="${exec.group}"/>
</java>
```

Code 9: Ant task for deployment using Java from Windows, Linux

The Java class is invoked in a `java` task with the `classpath` entries set to the required JAR files, viz. `ConfigManagerProxy.jar` and `com.ibm.mq.jar`. The class to be invoked is itself provided via yet another JAR file (`deployBAR-700.jar`) provided in the `classpath`. The class receives the name of the execution group and BAR file name as arguments and have been shown as Ant properties names.

10.4 Windows, Linux triggering z/OS batch job

```
<ftp server="${ipAddress}" userid="${userId}" password="${passWord}" remotedir="${remote.bar}" binary="true">
  <fileset file="${bar.filepath}"/>
</ftp>
<ftp server="${ipAddress}" userid="${userId}" password="${passWord}" remotedir="${targetDSN}" binary="false">
  <fileset file="${deploy.parms}"/>
</ftp>
```

Code 10: Ant task for triggering z/OS job from Windows, Linux

The first `ftp` task which sends the BAR file using binary transfer. In the second `ftp` task, a simple file with values of BAR file name and execution group is sent. This file will serve both as a trigger for a batch job and as an input to a step in the triggered batch job. This batch job step will parse values of BAR file name and execution group to create commands. These commands are then executed as batch job steps. The FTP credentials are shown as Ant properties.

10.5 Windows, Linux submitting z/OS batch job

```
<taskdef name="zJob" classname="com.ibm.wim.antTasks.ZJobSubmitTask">
  <classpath>
    <pathelement location="${tasks.jar.loc}/zJobSubmitTask.jar"/>
    <pathelement location="${tasks.jar.loc}/commons-net-3.0.1.jar"/>
  </classpath>
</taskdef>

<taskdef name="zJobRC" classname="com.ibm.wim.antTasks.ZJobRCTask">
  <classpath>
    <pathelement location="${tasks.jar.loc}/zJobSubmitTask.jar"/>
    <pathelement location="${tasks.jar.loc}/commons-net-3.0.1.jar"/>
  </classpath>
</taskdef>

<target name="run" description="Default">
  <echo message="Submitting job ..." />
  <zJob ipAddress="${ipAddress}" tsoUserID="${tsoUserID}" tsoPassword="${tsoPassword}"
  jobPath="${job.loc}" />

  <property file="${job.loc}.props"/>

  <echo message="Waiting 3 seconds ..." />
  <sleep seconds="3"/>

  <echo message="Getting result of submission ..." />
  <zJobRC ipAddress="${ipAddress}" tsoUserID="${tsoUserID}" tsoPassword="${tsoPassword}"
  jobID="${job.id}"/>

  <delete file="${job.id}"/>
</target>
```

Code 11: Ant tasks for submitting batch job and retrieving status from Windows and Linux

In this code snippet, we introduce two custom Ant tasks `zJob` and `zJobRC` that submit a job and retrieve the status of the submitted job respectively. Before submission, the job is modified with correct values. The FTP credentials are shown as Ant properties.

11 Adding external JARs and tasks for Ant

Certain Ant tasks have an external dependency on JAR files. For example, the `ftp` task requires Apache Commons-Net package and therefore, its JAR file should be available for Ant. This is set using the preferences of the client. Go to **Window** → **Preferences** → **Ant** → **Runtime ...**. Click on '**Add External JARs ...**' and navigate to the path of the location of the required JAR file and add the same. This step also needs to be performed to locate the JARs for custom Ant tasks.

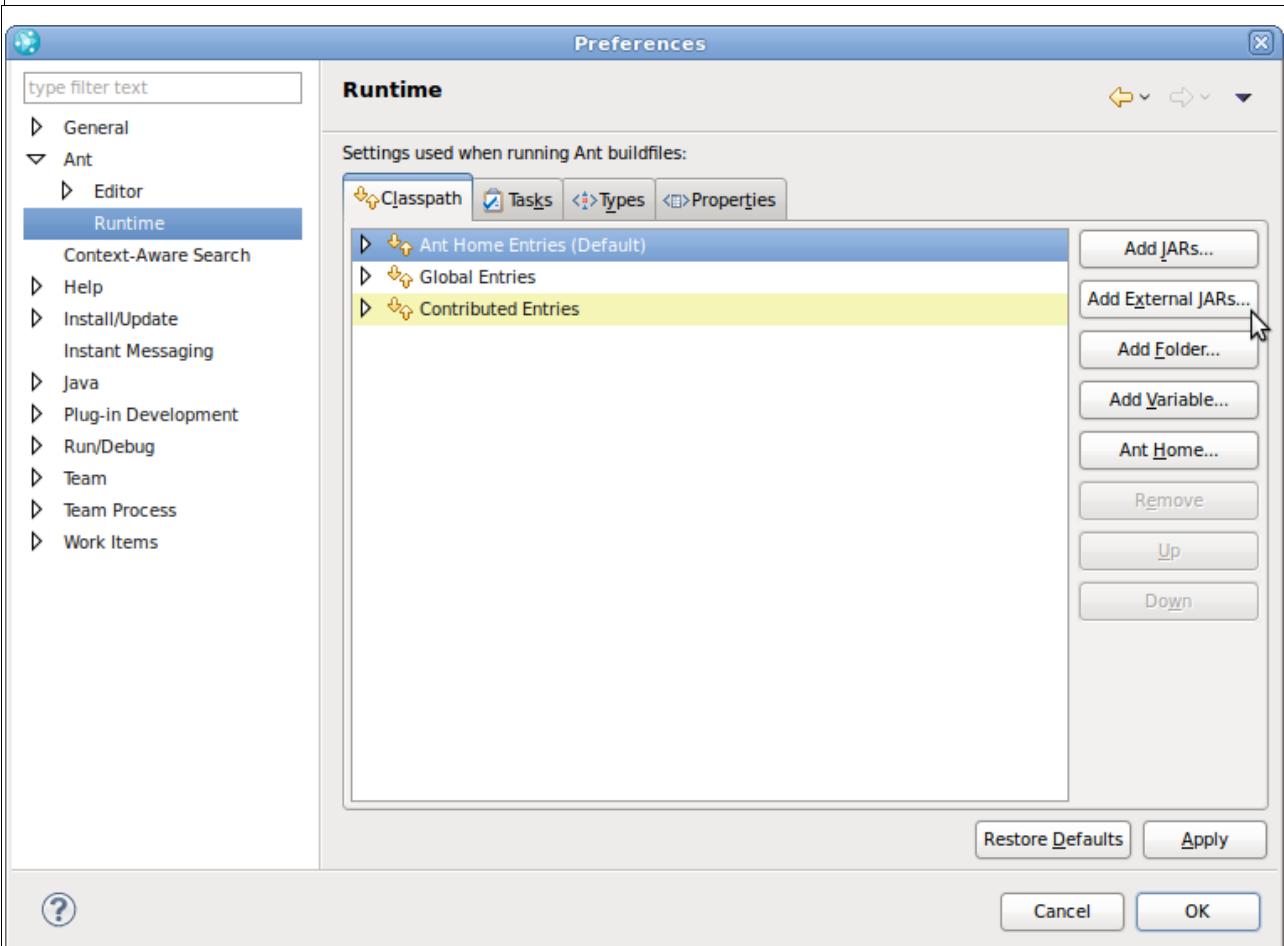


Fig 10: Changing Ant preferences to add external JARs

11.1 Add new Ant tasks

When using custom Ant tasks, the class name for the task should be set in `classpath` in the build file. However, when working in Eclipse client, the class name could be set so that Ant in the client treats the tasks as 'known'. To set the class name go to **Window** → **Preferences** → **Ant** → **Runtime ...**. Click on the **Tasks** tab and click on '**Add Task ...**'. In the resulting window, select location of JAR as added from the option of adding external JARs. Navigate to the class name and click OK.

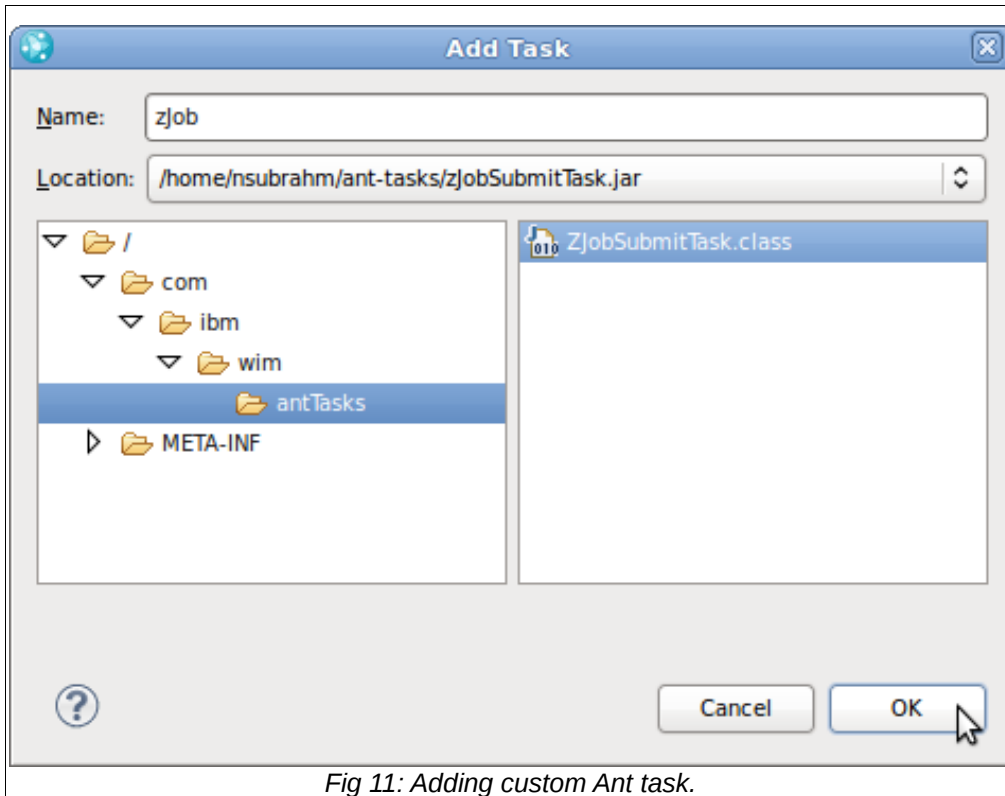


Fig 11: Adding custom Ant task.

Using the Ant build files

In this chapter, we will show how to use each of the Ant build files. These different build files cover the various deployment scenarios as discussed in Section [9.3](#). We will see each of the build file in detail, the one time set-up required for them and the changes required for each of them before initiating the build. Throughout all the sections in this chapter, it is assumed that you have a WebSphere Message Broker project (and its other referenced projects, if any) that was successfully built from the ToolKit.

The following build files are covered.

- [WMB ToolKit on Windows, using BAT file for deployment](#)
- [WMB ToolKit on Linux, using shell script for deployment](#)
- [WMB ToolKit on Windows, Linux using JAR file for deployment](#)
- [WMB ToolKit on Windows, Linux triggering batch job for z/OS deployment](#)
- [WMB ToolKit on Windows, Linux submitting batch job for z/OS deployment](#)

12 WMB ToolKit on Windows, using BAT file for deployment

In this section, we show the steps for automation of build, override and deployment of a BAR file. For this section, we will refer the `BuildFiles/Windows` folder.

12.1 *Build file and others*

This folder contains two Ant build files as follows.

1. `Sample-Windows-bat-build.xml` – Ant build file to create, override and deploy BAR file from WMB projects and share or check-in the WMB projects in RTC.
2. `Sample-RTC-deliver-build.xml` – Ant build file to deliver RTC change sets after (optionally) assigning comment, closing it and associating with RTC work item.

The properties files required are as follows.

1. `B1.props` – Pairs of name-value to connect to a broker on a system named B1.
2. `Override.props` – Pairs of name-value to override configurable properties in the generated BAR file.
3. `rtc.props` – Pairs of name-value to connect to a RTC server.
4. `Windows XP.props` – Pairs of name-value to the path of various executables invoked from this build file.

Other files required are as follows.

1. `deployBAR.bat` – Windows BAT file to deploy the generated BAR file to a broker.

12.2 *Description*

This build file is to be used from a Windows work-station when the deployment is to be done via a Windows BAT file. See section [9.1](#) for details. The targets in the build file and their brief description are as follows.

1. `init` – Display messages for the build to be initiated.
2. `buildBAR` – Generate BAR file from WMB projects (`-p` switch) and resources (`-o` switch). See Section [3](#).
3. `applyBAROverride` – Override configurable properties of BAR file using a properties file. This task will also write the current and changed values of configurable properties of BAR file. See Section [5](#).
4. `scmShare` – Share the WMB projects into RTC server. See Section [7](#).
5. `scmCheckin` – Check-in WMB projects into RTC server. See Section [7](#).
6. `deployBAR` – Deploy the BAR file – generated in `buildBAR` target – to the specified execution group in broker via a BAT file. See Section [6](#).
7. `cleanup` – Remove the workspace for building WMB projects.
8. `applyDeploy` – Run the `applyBAROverride`, `deployBAR` and `cleanup` targets sequentially on the generated BAR file.

Source Control with RTC and automation of build and deployment of WMB projects with Apache Ant

9. `allDeploy` – Run the `buildBAR`, `applyBAROverride`, `deployBAR` and `cleanup` targets sequentially.
10. `allShare` – Run the `buildBAR`, `applyBAROverride`, `scmShare`, `deployBAR` and `cleanup` targets sequentially.
11. `allCheckin` - Run the `buildBAR`, `applyBAROverride`, `scmCheckin`, `deployBAR` and `cleanup` targets sequentially.

12.3 Using the build file

In this section, we will discuss the steps to be followed to use this build file.

12.3.1 One-time set-up

The following steps are to be done only once. The home location is `C:` and is set in `filesystem.home` property in `Windows XP.props` file.

1. Create a folder `C:\Props` to save various properties files. The absolute path to this folder should tally with the Ant property `props.loc` in the build file.
2. Create a folder `C:\BAR` to save the generated BAR file. The absolute path to this folder should tally with the Ant property `local.bar.dir` in the build file.
3. Create a folder `C:\RTC-Build` to save the WMB projects that are to be shared or checked-in to RTC server. The absolute path to this folder should tally with the Ant property `rtc.home` in the build file.
4. Create a folder `C:\WMB-Build` to save the WMB projects that are to be used for generating the BAR file. The absolute path to this folder should tally with the Ant property `workspace.home` in the build file.
5. Create a folder `C:\BATFiles` to save the BAT file for deployment. The absolute path to this folder should tally with the Ant property `BAT.home` in the build file.
6. Copy the file `B1.props` to `C:\Props` and edit it with values suitable for the broker to be connected. The name of this file should be `target_system.props` where, `target_system` tallies with the Ant property `target.system` in the build file.
7. Copy the file `Override.props` to `C:\Props` and edit it with values suitable for the broker to be connected. The name of this file should tally with Ant property `override.config.props` in the build file.
8. Copy the file `rtc.props` to `C:\Props` and edit it with values suitable for the RTC server to be connected.
9. Copy the file `Windows XP.props` to `C:\Props`. The name of the file should tally the value of Ant built-in property `os.name`. On a Windows system, it might be `Windows XP`.

12.3.2 Steps before initiating the build

The following properties should have their value changed as shown below. These changes need to be done before every build.

1. `target.system` – A simple name to identify a system that hosts the broker.
2. `exec.group` – Name of an execution group on the broker to which the BAR file is to be

Source Control with RTC and automation of build and deployment of WMB projects with Apache Ant

deployed.

3. `current.config.props` – The path to a file where the current configurable properties of the generated BAR file are written to.
4. `changed.config.props` – The path to a file where the changed configurable properties of the generated BAR file are written to.
5. `override.config.props` – The path to a file where the override values of configurable properties of generated BAR file are provided. Using this file, configurable properties of generated BAR file are overridden.
6. `bar.filename` – The name of the BAR file to be generated from the WMB projects.
7. `rtc.repository.workspace` – The name of the remote RTC workspace with which the WMB projects would be shared or checked-in.
8. `rtc.repository.component` – The name of the component in the remote RTC workspace with which the WMB projects would be shared or checked-in.

The following changes need to be done for the following Ant targets.

1. `buildBAR` – In the `exec` task for generating the BAR file, add the names of WMB projects with the `-p` switch and the names of resources (message flow, `.classpath` or message set) to the `-o` switch.
2. `scmShare` – In the `exec` task for sharing WMB projects, add the names of WMB projects to be shared.
3. `scmCheckin` – In the `exec` task for sharing WMB projects, add the names of WMB projects to be checked-in.

12.3.3 Invoking Ant

Once all the changes – one time set-up and the changes for a specific project – are done, invoke Ant for this build file. To invoke Ant, see Section [8.2.1](#). By default, the `buildBAR` target is invoked that generates the BAR file. One may then selectively invoke other targets or one of the combined ones – `allDeploy`, `allShare` or `allCheckin`.

12.3.4 Notes

1. The file name `Windows XP.props` should tally with `${os.name}.props` derived from Ant.
2. The `applyBAROverride` target does not write its output to files when running on Windows platform. On Linux, the files are written to.
3. The `deployBAR.bat` file has the WMB version hard-coded.
4. The deployment syntax used is for deployment to remote brokers.
5. After sharing or checking-in the projects, the change sets have to be delivered. See Section [17](#).

13 WMB ToolKit on Linux, using shell script for deployment

For this section, we will refer the `BuildFiles/Linux` folder. This folder contains two Ant build files as follows.

1. `Sample-Linux-sh-build.xml` – Ant build file to create, override and deploy BAR file from WMB projects and share it check-in the WMB projects in RTC.
2. `Sample-RTC-deliver-build.xml` – Ant build file to deliver RTC change sets after (optionally) assigning comment, closing it and associating with RTC work item.

The properties files required are as follows.

1. `B1.props` – Pairs of name-value to connect to a broker on a system named B1.
2. `Override.props` – Pairs of name-value to override configurable properties in the generated BAR file.
3. `rtc.props` – Pairs of name-value to connect to a RTC server.
4. `Linux.props` – Pairs of name-value to the path of various executables invoked from this build file.

Other files required are as follows.

1. `deployBAR.sh` – Linux shell script file to deploy the generated BAR file to a broker.

13.1 Description

This build file is to be used from a Windows work-station when the deployment is to be done via a Windows BAT file. See section [9.1](#) for details. The targets in the build file and their brief description are as follows.

1. `init` – Display messages for the build to be initiated.
2. `buildBAR` – Generate BAR file from WMB projects (`-p` switch) and resources (`-o` switch). See Section [3](#).
3. `applyBAROverride` – Override configurable properties of BAR file using a properties file. This task will also write the current and changed values of configurable properties of BAR file. See Section [5](#).
4. `scmShare` – Share the WMB projects into RTC server. See Section [7](#).
5. `scmCheckin` – Check-in WMB projects into RTC server. See Section [7](#).
6. `deployBAR` – Deploy the BAR file – generated in `buildBAR` target – to the specified execution group in broker via a shell script file.
7. `cleanup` – Remove the workspace for building WMB projects.
8. `applyDeploy` – Run the `applyBAROverride`, `deployBAR` and `cleanup` targets sequentially on the generated BAR file.
9. `allDeploy` – Run the `buildBAR`, `applyBAROverride`, `deployBAR` and `cleanup` targets sequentially.
10. `allShare` – Run the `buildBAR`, `applyBAROverride`, `scmShare`, `deployBAR` and `cleanup` targets sequentially.

11. `allCheckin` - Run the `buildBAR`, `applyBAROverride`, `scmCheckin`, `deployBAR` and `cleanup` targets sequentially.

13.2 Using the build file

In this section, we will discuss the steps to be followed to use this build file.

13.2.1 One-time set-up

The following steps are to be done only once. For this section, `$HOME` stands for home folder of a user. The `$HOME` is also the value set in `filesystem.home` property in `Linux.props` file.

1. Create a folder `$HOME/Props` to save various properties files. The absolute path to this folder should tally with the Ant property `props.loc` in the build file.
2. Create a folder `$HOME/BAR` to save the generated BAR file. The absolute path to this folder should tally with the Ant property `local.bar.dir` in the build file.
3. Create a folder `$HOME/RTC-Build` to save the WMB projects that are to be shared or checked-in to RTC server. The absolute path to this folder should tally with the Ant property `rtc.home` in the build file.
4. Create a folder `$HOME/WMB-Build` to save the WMB projects that are to be used for generating the BAR file. The absolute path to this folder should tally with the Ant property `workspace.home` in the build file.
5. Create a folder `$HOME/Code` to save the shell script file for deployment. The absolute path to this folder should tally with the Ant property `sh.home` in the build file.
6. Copy the file `B1.props` to `$HOME/Props` and edit it with values suitable for the broker to be connected. The name of this file should be `target_system.props` where, `target_system` tallies with the Ant property `target.system` in the build file.
7. Copy the file `Override.props` to `$HOME/Props` and edit it with values suitable for the broker to be connected. The name of this file should tally with Ant property `override.config.props` in the build file.
8. Copy the file `rtc.props` to `$HOME/Props` and edit it with values suitable for the RTC server to be connected.
9. Copy the file `Linux.props` to `$HOME/Props`. The name of the file should not be changed as the name will be obtained from the Ant built-in property `os.name`.

13.2.2 Steps before initiating the build

The following properties should have their value changed as shown below. These changes need to be done before every build.

1. `target.system` – A simple name to identify a system that hosts the broker.
2. `exec.group` – Name of an execution group on the broker to which the BAR file is to be deployed.
3. `current.config.props` – The path to a file where the current configurable properties of the generated BAR file are written to.
4. `changed.config.props` – The path to a file where the changed configurable properties

Source Control with RTC and automation of build and deployment of WMB projects with Apache Ant

of the generated BAR file are written to.

5. `override.config.props` – The path to a file where the override values of configurable properties of generated BAR file are provided. Using this file, configurable properties of generated BAR file are overridden.
6. `bar.filename` – The name of the BAR file to be generated from the WMB projects.
7. `rtc.repository.workspace` – The name of the remote RTC workspace with which the WMB projects would be shared or checked-in.
8. `rtc.repository.component` – The name of the component in the remote RTC workspace with which the WMB projects would be shared or checked-in.

The following changes need to be done for the following Ant targets.

1. `buildBAR` – In the `exec` task for generating the BAR file, add the names of WMB projects with the `-p` switch and the names of resources (message flow, `.classpath` or message set) to the `-o` switch.
2. `scmShare` – In the `exec` task for sharing WMB projects, add the names of WMB projects to be shared.
3. `scmCheckin` – In the `exec` task for sharing WMB projects, add the names of WMB projects to be checked-in.

13.2.3 Invoking Ant

Once all the changes – one time set-up and the changes for a specific project – are done, invoke Ant for this build file. To invoke Ant, see Section [8.2.1](#). By default, the `buildBAR` target is invoked that generates the BAR file. One may then selectively invoke other targets or one of the combined ones – `allDeploy`, `allShare` or `allCheckin`.

13.2.4 Notes

1. The file name `Linux.props` should tally with `${os.name}.props` derived from Ant.
2. The `deployBAR.sh` file has the WMB version hard-coded.
3. The deployment syntax used is for deployment to remote brokers.
4. After sharing or checking-in the projects, the change sets have to be delivered. See Section [17](#).

14 WMB ToolKit on Windows, Linux using JAR file for deployment

For this section, we will refer the `BuildFiles/Windows-Linux` folder. This folder contains two Ant build files as follows.

1. `Sample-WMB-RTC-build.xml` – Ant build file to create, override and deploy BAR file from WMB projects and share it check-in the WMB projects in RTC.
2. `Sample-RTC-deliver-build.xml` – Ant build file to deliver RTC change sets after (optionally) assigning comment, closing it and associating with RTC work item.

The properties files required are as follows.

1. `B1.props` – Pairs of name-value to connect to a broker on a system named B1.
2. `Override.props` – Pairs of name-value to override configurable properties in the generated BAR file.
3. `rtc.props` – Pairs of name-value to connect to a RTC server.
4. `Windows XP.props` – Pairs of name-value to the path of various executables on Windows platform invoked from this build file.
5. `Linux.props` – Pairs of name-value to the path of various executables on Linux platform invoked from this build file.

Other files required are as follows.

1. `deployBAR-700.jar` – JAR file to deploy the generated BAR file to a broker.

14.1 Description

This build file is to be used from a Windows work-station when the deployment is to be done via a Windows BAT file. See section [9.1](#) for details. The targets in the build file and their brief description are as follows.

1. `init` – Display messages for the build to be initiated.
2. `buildBAR` – Generate BAR file from WMB projects (`-p` switch) and resources (`-o` switch). See Section [3](#).
3. `applyBAROverride` – Override configurable properties of BAR file using a properties file. This task will also write the current and changed values of configurable properties of BAR file. See Section [5](#).
4. `scmShare` – Share the WMB projects into RTC server. See Section [7](#).
5. `scmCheckin` – Check-in WMB projects into RTC server. See Section [7](#).
6. `deployBAR` – Deploy the BAR file – generated in `buildBAR` target – to the specified execution group in broker via a BAT file.
7. `cleanup` – Remove the workspace for building WMB projects.
8. `applyDeploy` – Run the `applyBAROverride` and `deployBAR` targets sequentially on the generated BAR file.
9. `allDeploy` – Run the `buildBAR`, `applyBAROverride`, `deployBAR` and `cleanup` targets sequentially.

10. allShare – Run the buildBAR, applyBAROverride, scmShare, deployBAR and cleanup targets sequentially.
11. allCheckin - Run the buildBAR, applyBAROverride, scmCheckin, deployBAR and cleanup targets sequentially.

14.2 Using the build file

In this section, we will discuss the steps to be followed to use this build file.

14.2.1 One-time set-up

The following items are to be done only once. For this section, \$HOME will mean C: on Windows platform and home directory of a user on Linux platform. It is also the value set in filesystem.home property in Linux.props or Windows XP.props files. The file path separator will be as applicable on Windows or Linux.

1. Create a folder \$HOME/Props to save various properties files. The absolute path to this folder should tally with the Ant property props.loc in the build file.
2. Create a folder \$HOME/BAR to save the generated BAR file. The absolute path to this folder should tally with the Ant property local.bar.dir in the build file.
3. Create a folder \$HOME/JAR to hold the JAR file for deployment. The absolute path to this folder should tally with the Ant property deploy.jar in the build file.
4. Create a folder \$HOME/RTC-Build to save the WMB projects that are to be shared or checked-in to RTC server. The absolute path to this folder should tally with the Ant property rtc.home in the build file.
5. Create a folder \$HOME/WMB-Build to save the WMB projects that are to be used for generating the BAR file. The absolute path to this folder should tally with the Ant property workspace.home in the build file.
6. Copy the file B1.props to \$HOME/Props and edit it with values suitable for the broker to be connected. The name of this file should be target_system.props where, target_system tallies with the Ant property target.system in the build file.
7. Copy the file Override.props to \$HOME/Props and edit it with values suitable for the broker to be connected. The name of this file should tally with Ant property override.config.props in the build file.
8. Copy the file rtc.props to \$HOME/Props and edit it with values suitable for the RTC server to be connected.
9. Copy the file Windows XP.props or Linux.props to \$HOME/Props. The name of the files should not be changed as the name will be obtained from the Ant built-in property os.name.
10. Copy the file deployBAR-700.jar to \$HOME/JAR. The name of the file should not be changed as this name would be searched for in the java task for deployment in deployBAR target.

14.2.2 Steps before initiating the build

The following properties should have their value changed as shown below. These changes need

to be done before every build.

1. `target.system` – A simple name to identify a system that hosts the broker.
2. `exec.group` – Name of an execution group on the broker to which the BAR file is to be deployed.
3. `current.config.props` – The path to a file where the current configurable properties of the generated BAR file are written to.
4. `changed.config.props` – The path to a file where the changed configurable properties of the generated BAR file are written to.
5. `override.config.props` – The path to a file where the override values of configurable properties of generated BAR file are provided. Using this file, configurable properties of generated BAR file are overridden.
6. `bar.filename` – The name of the BAR file to be generated from the WMB projects.
7. `rtc.repository.workspace` – The name of the remote RTC workspace with which the WMB projects would be shared or checked-in.
8. `rtc.repository.component` – The name of the component in the remote RTC workspace with which the WMB projects would be shared or checked-in.

The following changes need to be done for the following Ant targets.

1. `buildBAR` – In the `exec` task for generating the BAR file, add the names of WMB projects with the `-p` switch and the names of resources (message flow, `.classpath` or message set) to the `-o` switch.
2. `scmShare` – In the `exec` task for sharing WMB projects, add the names of WMB projects to be shared.
3. `scmCheckin` – In the `exec` task for sharing WMB projects, add the names of WMB projects to be checked-in.

14.2.3 Invoking Ant

Once all the changes – one time set-up and the changes for a specific project – are done, invoke Ant for this build file. To invoke Ant, see Section [8.2.1](#). By default, the `buildBAR` target is invoked that generates the BAR file. One may then selectively invoke other targets or one of the combined ones – `allDeploy`, `allShare` or `allCheckin`.

14.2.4 Notes

1. The `applyBAROverride` target does not write its output to files when running on Windows platform. On Linux, the files are written to.
2. The file name `Windows.XP.props` or `Linux.props` should tally with `os.name` derived from Ant.
3. After sharing or checking-in the projects, the change sets have to be delivered. See Section [17](#).

15 WMB ToolKit on Windows, Linux triggering batch job for z/OS deployment

For this section, we will refer the `BuildFiles/zOS` folder. This folder contains two Ant build files as follows.

1. `Sample-zOS-WMB-RTC-build.xml` – Ant build file to create, override and deploy BAR file from WMB projects and share it check-in the WMB projects in RTC.
2. `Sample-RTC-deliver-build.xml` – Ant build file to deliver RTC change sets after (optionally) assigning comment, closing it and associating with RTC work item.

The properties files required are as follows.

1. `B1.props` – Pairs of name-value to connect to a broker on a system named B1.
2. `B1-ftp.props` – Pairs of name-value to connect to FTP server on a system named B1.
3. `Override.props` – Pairs of name-value to override configurable properties in the generated BAR file.
4. `rtc.props` – Pairs of name-value to connect to a RTC server.
5. `Windows XP.props` – Pairs of name-value to the path of various executables on Windows invoked from this build file.
6. `Linux.props` – Pairs of name-value to the path of various executables on Linux invoked from this build file.

Other files required are as follows.

1. `deploy.jcl` – A z/OS batch job in a z/OS partitioned dataset (PDS) member that would be submitted by a job scheduler via a dataset trigger.
2. `deploy.parms` – A file with keys and their values that are dynamically changed in every build. The changed file is sent via FTP to a z/OS dataset to trigger a batch job via a scheduler.

15.1 Description

This build file is to be used from a Windows work-station when the deployment is to be done via a Windows BAT file. See section [9.1](#) for details. The targets in the build file and their brief description are as follows.

1. `init` – Display messages for the build to be initiated.
2. `buildBAR` – Generate BAR file from WMB projects (`-p` switch) and resources (`-o` switch). See Section [3](#).
3. `scmShare` – Share the WMB projects into RTC server. See Section [7](#).
4. `scmCheckin` – Check-in WMB projects into RTC server.
5. `invokeDeploy` – Send a file via FTP with keys and values separated by commas to a z/OS dataset that will trigger a z/OS batch job. The file `deploy.parms` is changed with values to `deploy.parms.changed` and sent via FTP to a z/OS dataset that triggers a batch job. Before sending this file, the BAR file and override properties file are sent to z/OS via FTP. See Section [9.2](#).

Source Control with RTC and automation of build and deployment of WMB projects with Apache Ant

6. `cleanup` – Remove the workspace for building WMB projects.
7. `applyDeploy` – Run the `invokeDeploy` and `cleanup` targets sequentially.
8. `allDeploy` – Run the `buildBAR`, `invokeDeploy` and `cleanup` targets sequentially.
9. `allShare` – Run the `buildBAR`, `invokeDeploy`, `scmShare` and `cleanUp` targets sequentially.
10. `allCheckin` - Run the `buildBAR`, `invokeDeploy`, `scmCheckin` and `cleanup` targets sequentially.

15.2 Using the build file

In this section, we will discuss the steps to be followed to use this build file. In this section, `$HOME` stands for `C:` on Windows or the home directory of a user on Linux work-station. Similarly, `$USS-HOME` stands for the home directory of a user on USS file-system.

15.2.1 One-time set-up

The following items are to be done once only. For this section, `$HOME` will mean `C:` on Windows platform and home directory of a user on Linux platform. It is also the value set in `filesystem.home` property in `Linux.props` or `Windows XP.props` files. The file path separator will be as applicable on Windows or Linux.

1. Create a folder `$HOME/Props` to save various properties files. The absolute path to this folder should tally with the Ant property `props.loc` in the build file.
2. Create a folder `$HOME/BAR` to save the generated BAR file. The absolute path to this folder should tally with the Ant property `local.bar.dir` in the build file.
3. Create a folder `$HOME/RTC-Build` to save the WMB projects that are to be shared or checked-in to RTC server. The absolute path to this folder should tally with the Ant property `rtc.home` in the build file.
4. Create a folder `$HOME/WMB-Build` to save the WMB projects that are to be used for generating the BAR file. The absolute path to this folder should tally with the Ant property `workspace.home` in the build file.
5. Create a folder `$USS-HOME/BAR` on Unix System Services (USS) to save the BAR generated on the work-station.
6. Create a folder `$USS-HOME/Props` on Unix System Services (USS) to save the properties file for applying overrides.
7. Set the value of `trigger.dsn` Ant property to the name of the z/OS dataset that will trigger the batch job. Be sure to escape the quotes in the dataset name, if applicable.
8. Copy the file `B1.props` to `$HOME/Props` and edit it with values suitable for the broker to be connected. The name of this file should be `target_system.props` where, `target_system` tallies with the Ant property `target.system` in the build file.
9. Copy the file `Override.props` to `$HOME/Props` and edit it with values suitable for the broker to be connected. The name of this file should tally with Ant property `override.config.props` in the build file.
10. Copy the file `rtc.props` to `$HOME/Props` and edit it with values suitable for the RTC server to be connected.

Source Control with RTC and automation of build and deployment of WMB projects with Apache Ant

11. Copy the file `Windows.XP.props` to `$HOME/Props`. The name of the file should not be changed as the name will be obtained from the Ant built-in property `os.name`.
12. Copy the file `Linux.props` to `$HOME/Props`. The name of the file should not be changed as the name will be obtained from the Ant built-in property `os.name`.

15.2.2 Steps before initiating the build

The following properties should have their value changed as shown below. These changes need to be done before every build.

1. `target.system` – A simple name to identify a system that hosts the broker. This name is used for couple of properties files making it easy to differentiate properties for connectivity between different systems.
2. `exec.group` – Name of an execution group on the broker to which the BAR file is to be deployed.
3. `current.config.props` – The path to a file where the current configurable properties of the generated BAR file are written to.
4. `changed.config.props` – The path to a file where the changed configurable properties of the generated BAR file are written to.
5. `override.config.props` – The path to a file where the override values of configurable properties of generated BAR file are provided. Using this file, configurable properties of generated BAR file are overridden.
6. `bar.filename` – The name of the BAR file to be generated from the WMB projects.
7. `rtc.repository.workspace` – The name of the remote RTC workspace with which the WMB projects would be shared or checked-in.
8. `rtc.repository.component` – The name of the component in the remote RTC workspace with which the WMB projects would be shared or checked-in.

The following changes need to be done for the following Ant targets.

1. `buildBAR` – In the `exec` task for generating the BAR file, add the names of WMB projects with the `-p` switch and the names of resources (message flow, `.classpath` or message set) to the `-o` switch.
2. `scmShare` – In the `exec` task for sharing WMB projects, add the names of WMB projects to be shared.
3. `scmCheckin` – In the `exec` task for sharing WMB projects, add the names of WMB projects to be checked-in.

15.2.3 Invoking Ant

Once all the changes – one time set-up and the changes for a specific project – are done, invoke Ant for this build file. To invoke Ant, see Section [8.2.1](#). By default, the `buildBAR` target is invoked that generates the BAR file. One may then selectively invoke other targets or one of the combined ones – `allDeploy`, `allShare` or `allCheckin`.

15.2.4 Notes

1. To deliver the change sets, see Section [17](#).

16 WMB ToolKit on Windows, Linux submitting batch job for z/OS deployment

For this section, we will refer the `BuildFiles/CustomAnt` folder. This folder contains two Ant build files as follows.

1. `Sample-CustomAnt-WMB-RTC-build.xml` – Ant build file to create, override and deploy BAR file from WMB projects and share or check-in the WMB projects in RTC.
2. `Sample-RTC-deliver-build.xml` – Ant build file to deliver RTC change sets after (optionally) assigning comment, closing it and associating with RTC work item.

The properties files required are as follows.

1. `B1-ftp.props` – Pairs of name-value to connect to FTP server on a system named B1.
2. `Override.props` – Pairs of name-value to override configurable properties in the generated BAR file.
3. `rtc.props` – Pairs of name-value to connect to a RTC server.
4. `Windows XP.props` – Pairs of name-value to the path of various executables on Windows invoked from this build file.
5. `Linux.props` – Pairs of name-value to the path of various executables on Linux invoked from this build file.

Other files required are as follows.

1. `deploy.jcl` – A z/OS batch job as a file on the local file-system that would be submitted.
2. `commons-net-3.0.1.jar` – A JAR file required by custom Ant tasks.
3. `zjobRCTask.jar` – A JAR file for the custom Ant task `zJobRC` that checks for the return code of the submitted batch job.
4. `zjobSubmitTask.jar` – A JAR file for the custom Ant task `zJob` that submits a batch job.

16.1 Description

This build file is to be used from a Windows or Linux work-stations when the deployment is to be done using custom Ant tasks. See section [9.1](#) for details. The targets in the build file and their brief description are as follows.

1. `init` – Display messages for the build to be initiated.
2. `buildBAR` – Generate BAR file from WMB projects (`-p` switch) and resources (`-o` switch). See Section [3](#).
3. `scmShare` – Share the WMB projects into RTC server. See Section [7](#).
4. `scmCheckin` – Check-in WMB projects into RTC server. See Section [7](#).
5. `prepareJob` – Prepare the batch job for submission by applying edits to file `deploy.jcl`. See Section [9.2](#).
6. `submitJob` – Submit the prepared job (created in `prepareJob` target) using the `zJob` custom Ant task. See Section [9.2](#).

Source Control with RTC and automation of build and deployment of WMB projects with Apache Ant

7. `checkJob` – Check the return code of the job submitted in `submitJob` target using `zJobRC` custom Ant task. See Section [9.2](#).
8. `cleanup` – Remove the workspace for building WMB projects.
9. `applyDeploy` – Run the `applyBAROverride` and `deployBAR` targets sequentially on the generated BAR file.
10. `allDeploy` – Run the `buildBAR`, `applyBAROverride`, `deployBAR` and `cleanUp` targets sequentially.
11. `allShare` – Run the `buildBAR`, `applyBAROverride`, `scmShare`, `deployBAR` and `cleanUp` targets sequentially.
12. `allCheckin` – Run the `buildBAR`, `applyBAROverride`, `scmCheckin`, `deployBAR` and `cleanUp` targets sequentially.

16.2 Using the build file

In this section, we will discuss the steps to be followed to use this build file. In this section, `$HOME` stands for `C:` on Windows or the home directory of a user on Linux work-station. Similarly, `$USS-HOME` stands for the home directory of a user on USS file-system.

16.2.1 One-time set-up

The following items are to be done once only. For this section, `$HOME` will mean `C:` on Windows platform and home directory of a user on Linux platform. It is also the value set in `filesystem.home` property in `Linux.props` or `Windows XP.props` files. The file path separator will be as applicable on Windows or Linux.

1. Create a folder `$HOME/Props` to save various properties files. The absolute path to this folder should tally with the Ant property `props.loc` in the build file.
2. Create a folder `$HOME/BAR` to save the generated BAR file. The absolute path to this folder should tally with the Ant property `local.bar.dir` in the build file.
3. Create a folder `$HOME/RTC-Build` to save the WMB projects that are to be shared or checked-in to RTC server. The absolute path to this folder should tally with the Ant property `rtc.home` in the build file.
4. Create a folder `$HOME/WMB-Build` to save the WMB projects that are to be used for generating the BAR file. The absolute path to this folder should tally with the Ant property `workspace.home` in the build file.
5. Create a folder `$USS-HOME/BAR` on Unix System Services (USS) to save the BAR generated on the work-station.
6. Create a folder `$USS-HOME/Props` on Unix System Services (USS) to save the properties file for applying overrides.
7. Copy the file `B1.props` to `$HOME/Props` and edit it with values suitable for the broker to be connected. The name of this file should be `target_system.props` where, `target_system` tallies with the Ant property `target.system` in the build file.
8. Copy the file `Override.props` to `$HOME/Props` and edit it with values suitable for the broker to be connected. The name of this file should tally with Ant property `override.config.props` in the build file.

Source Control with RTC and automation of build and deployment of WMB projects with Apache Ant

9. Copy the file `rtc.props` to `$HOME/Props` and edit it with values suitable for the RTC server to be connected.
10. Copy the file `Windows XP.props` to `$HOME/Props`. The name of the file should not be changed as the name will be obtained from the Ant built-in property `os.name`.
11. Copy the file `Linux.props` to `$HOME/Props`. The name of the file should not be changed as the name will be obtained from the Ant built-in property `os.name`.
12. Copy the file `deploy.jcl` to any location on file-system and edit it as per requirements of z/OS machine. The absolute path to this file should tally with `file` attribute in `copy` task of the `prepareJob` target.

16.2.2 Steps before initiating the build

The following properties should have their value changed as shown below. These changes need to be done before every build.

1. `target.system` – A simple name to identify a system that hosts the broker. This name is used for couple of properties files making it easy to differentiate properties for connectivity between different systems.
2. `exec.group` – Name of an execution group on the broker to which the BAR file is to be deployed.
3. `current.config.props` – The path to a file where the current configurable properties of the generated BAR file are written to.
4. `changed.config.props` – The path to a file where the changed configurable properties of the generated BAR file are written to.
5. `override.config.props` – The path to a file where the override values of configurable properties of generated BAR file are provided. Using this file, configurable properties of generated BAR file are overridden.
6. `bar.filename` – The name of the BAR file to be generated from the WMB projects.
7. `rtc.repository.workspace` – The name of the remote RTC workspace with which the WMB projects would be shared or checked-in.
8. `rtc.repository.component` – The name of the component in the remote RTC workspace with which the WMB projects would be shared or checked-in.

The following changes need to be done for the following Ant targets.

1. `buildBAR` – In the `exec` task for generating the BAR file, add the names of WMB projects with the `-p` switch and the names of resources (message flow, `.classpath` or message set) to the `-o` switch.
2. `scmShare` – In the `exec` task for sharing WMB projects, add the names of WMB projects to be shared.
3. `scmCheckin` – In the `exec` task for sharing WMB projects, add the names of WMB projects to be checked-in.

16.2.3 Invoking Ant

Once all the changes – one time set-up and the changes for a specific project – are done, invoke Ant for this build file. To add the custom tasks, see Section [11](#). To invoke Ant, see Section [8.2.1](#).

By default, the `buildBAR` target is invoked that generates the BAR file. One may then selectively invoke other targets or one of the combined ones – `allDeploy`, `allShare` or `allCheckin`.

16.2.4 Notes

1. In this build file, the `zJobRC` task will look for the status of the submitted job by looking for the JOBID of the submitted job. This JOBID is written to a file referred to by `job.id` Ant property.
2. After sharing or checking-in the projects, the change sets have to be delivered. See [Section 17](#).

17 Delivering change sets to RTC server

Once a WMB project has been shared or checked-in to the RTC remote workspace, then the changes will remain pending until they are actually delivered to the stream. While one can directly deliver the change set from the GUI of RTC Eclipse client, it can also be automated in an Ant build file. In this section, we will discuss this automation as an Ant build file. However, before that, we first propose the following sequence of RTC activities.

1. Set comment on the change set.
2. Associate the change set with a work item.
3. Mark the change set complete.
4. Deliver the work item (change set).

For this section, we are referring to the `Sample-RTC-deliver.xml` file. The required properties file are as follows.

1. `rtc.props` – Pairs of name-value to connect to RTC server.
2. `Windows XP.props` – Pairs of name-value to the path of various executables on Windows invoked from this build file.
3. `Linux.props` – Pairs of name-value to the path of various executables on Linux invoked from this build file.

17.1 Description

This build file is to be used from a Windows or Linux work-stations when change sets are to be delivered to Rational Team Concert server. This build file has the following targets and in this order.

1. `init`. This task shows the complete path of the build file and the time-stamp when the build was initiated.
2. `setComment`. This task applies a comment to the change-set to which a WMB or a WTX project was checked-in to. The format of the comment will be user name (derived from the built-in Ant variable `user.name`), work item number, change-set alias and current time-stamp separated by the underscore character.
3. `assocChangeSet`. This task associates the change-set with a work item number. This task will assume importance in cases where there is a policy to deliver only those change sets that have an associated work item.
4. `completeChangeSet`. This task marks the change-set complete.
5. `scmDeliverWI`. This task delivers changes at the level of a work-item.
6. `scmDeliverCS`. This task delivers changes at the level of an individual change-set.

17.2 Using the build file

In this section, we will discuss the steps to be followed to use this build file. In this section, `$HOME` stands for C: on Windows or the home directory of a user on Linux work-station.

17.2.1 One time set-up

The following steps need to be done only once.

1. Create a folder `$HOME/Props` to save various properties files. The absolute path to this folder should tally with the Ant property `props.loc` in the build file.
2. Create a folder `$HOME/RTC-Build` to save the WMB projects that are to be shared or checked-in to RTC server. The absolute path to this folder should tally with the Ant property `rtc.home` in the build file.

17.2.2 Steps before initiating the build

The following properties should have their value changed as shown below. These changes need to be done before every build.

1. `changeset.alias` – The four digit number for the change to be delivered.
2. `work.item` – The RTC work item to be associated with the change set aliases.

17.2.3 Invoking Ant

Once all the changes – one time set-up and the changes for a specific project – are done, invoke Ant for this build file. To invoke Ant, see Section [8.2.1](#). By default, the `allDeliverWI` target is invoked that delivers the work-item and its associated change-set.

17.2.4 Notes

1. The current design of RTC does not allow for delivery of a work item that has multiple change sets associated with it. (See defect raised in jazz.net site – [181229](#)).

Conclusion

In conclusion, this SupportPac describes the usage of Ant build files to automate the build, override and deploy Broker Archive files generated from WebSphere Message Broker projects. This SupportPac also describes an automated way to share the projects in Rational Team Concert server. In this chapter, we present a life-cycle of WMB projects from the perspective of deployment to a broker and delivery to RTC server.

- Life-cycle of WMB project – deployment and source control

18 Life-cycle of WMB project – deployment and source control

The usage of the SupportPac is expected to be aligned with the life-cycle shown in the Fig 13 below.

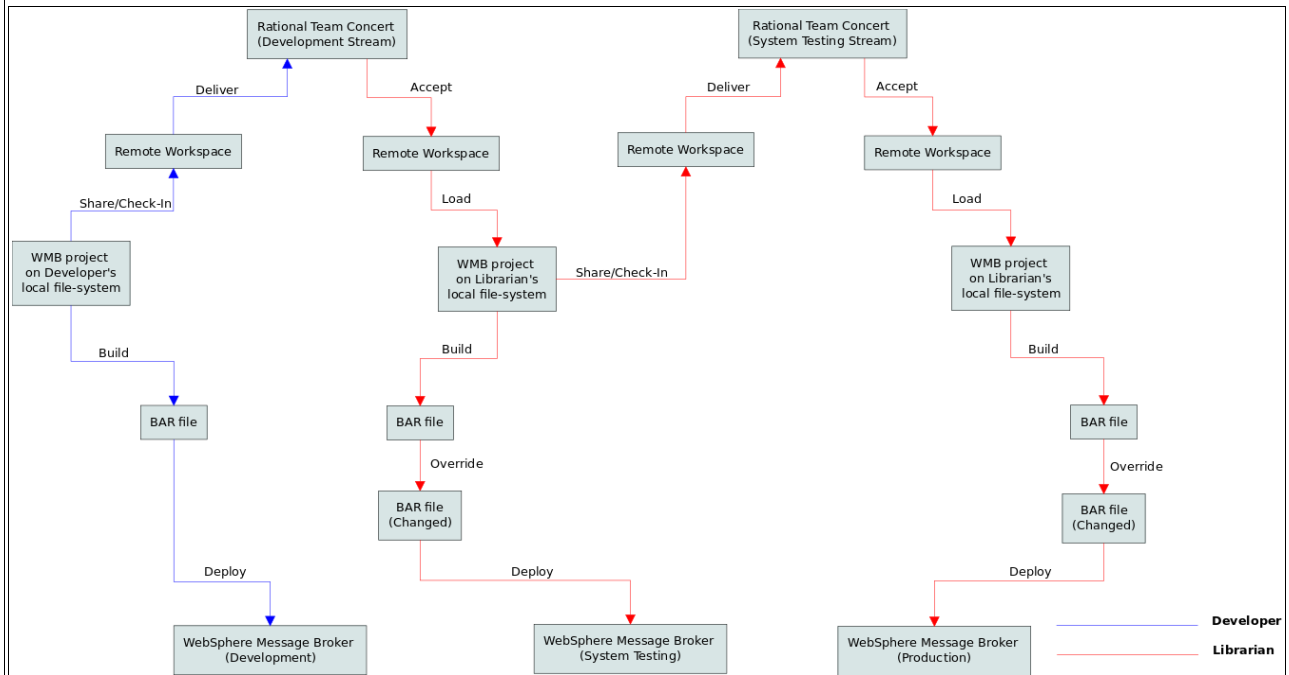


Fig 12 Illustrative life-cycle of WMB project – deployment and source control

The life-cycle is briefly explained below.

1. Developer finishes the development of WMB project and creates a new Ant build file copied from one of the samples provided. The developer also prepares the properties file of different target environment for overriding the configurable properties.
2. With this build file, a developer can build and deploy the BAR file (from the finished WMB project) to the development broker in development region. Also, the developer may deliver to the RTC development stream.
3. The librarian will accept and load the changes to his/her local file-system from the RTC development stream. Then, with the Ant build file checked-in by the developer, the librarian generates the BAR file afresh. The configurable properties of this BAR file will be overridden for system testing region using the properties file checked-in by the developer. Finally, the BAR file is deployed to broker in system testing region.
4. Once the librarian has accepted and loaded the changes to his/her local file-system, the WMB project may also be delivered to the RTC system testing stream using the Ant build file checked-in by the developer.
5. The steps 3 and 4 are then repeated while moving projects and BAR file to production stream and broker.

19 Bibliography

1. [WebSphere Message Broker v7 Infocenter](#) – WMB documentation.
2. [Rational Team Concert v3 Infocenter](#) – RTC documentation.
3. [Apache Ant](#) – Ant build tool documentation.
4. [Java Development with Ant](#) – Nice book describing Ant extensions in Java.
5. [Apache Commons](#) – collection of reusable Java components.
6. [Jazz Source Control FAQ](#) – jazz.net article on FAQ related to source control.
7. [Getting Started with Jazz Source Control](#) – jazz.net article on getting started with source control.
8. [Getting started with Work Items in Rational Team Concert](#) – jazz.net article describing RTC work items.

20 Author biography

Nagesh Subrahmanyam is an Advisory System Analyst with IBM India Pvt Ltd. He has over a decade of experience spanning z/OS, WebSphere, XML and open source technologies. He can be contacted at nsubrahm@in.ibm.com.