Rational Team Concert for System z
Beta Program

Installation and User's Guide

Version 2.0 Beta 2

Rational Team Concert for System z
Beta Program

Installation and User's Guide

Version 2.0 Beta 2

Note
Before using this information and the product it supports, read the information in "Notices" on page 101.

This edition applies to the beta 2 version of IBM Rational Team Concert for System z V2.0 (product number 5724-V82).

Contents

Introduction to Rational Team Concert for System z beta

IBM Rational Team Concert for System z is a collaborative software delivery environment that teams can use to simplify, automate, and govern application development on System z. This beta program is intended to provide early access to the product while it is being developed.

Rational Team Concert for System z includes the following features:

- Specialized support for traditional language artifacts such as COBOL
- Integrated collaborative software delivery lifecycle tools for System z development, including source control, change management, build and process management
- Support for multitier software development and application modernization efforts
- Native z file system support
- System z artifact build script generation and management

Rational Team Concert for System z extends the base Rational Team Concert functions in the following ways:

- Supports running the Jazz Team Server in a System z environment.
- Supports using DB2 for System z as the Jazz Team Server back-end repository.
- Extends the core Build System Toolkit to include a System z Jazz Gateway that allows external software configuration management systems (SCMs) to query Jazz-based work items.
- Provides support for a System z project structure and System z build support through the use of Ant scripting and the Rational Build Agent.

Based on the open and extensible Jazz platform, Rational Team Concert for System z empowers software teams to collaborate in real time in the context of specific roles and processes. The product integrates source control, work items, build management, dashboards, and process support into a single solution for System z.

About this beta guide

This guide provides information about installing, configuring, and using the beta version of Rational Team Concert for System z.

The information in this guide supplements the Rational Team Concert online help available from the help menu of the installed client and at http://publib.boulder.ibm.com/infocenter/rtc/v1r0m1/index.jsp.

Note: You can find additional information about Rational Team Concert when you register at jazz.net.

This guide provides instructions for installing the following Rational Team Concert for System z beta components:

- Jazz Team Server for System z
- Rational Team Concert for System z client
- Build System Toolkit for System z

- Rational Build Agent

After you complete the installation, you can work with Rational Team Concert basic tutorials or explore the Rational Team Concert for System z beta user scenarios, which are included in this guide.

Chapter 1. Installing and configuring Rational Team Concert for System z

The installation method for Beta 2 is IBM's Installation Manager for Windows, Linux, and Linux on System z and SMP/E for z/OS. Download the appropriate package for the target platform from the RTCz beta jazz.net download site and unzip it onto the target machine.

The Rational Team Concert for System z beta package includes several components specifically designed for application development on System z. You can download all Rational Team Concert for System z beta files from the beta Web site. The Rational Team Concert for System z beta package includes the following components:

- Jazz Team Server for System z
- Rational Team Concert for System z client
- Build System Toolkit for System z (includes the System z Jazz Gateway and the Jazz Build Engine)
- Rational Build Agent

To install and run the beta program, go to the beta Web site at http://jazz.net/downloads/rational-team-concert/betas/z2.0Beta2 and follow the instructions provided on the Getting started tab.

All Rational Team Concert for System z components that can be installed on the System z machine are contained within an SMP/E package. This package is contained within the download file RTCz-SMPE-2.0Beta2.zip. This file also contains the Program Directory for the SMP/E package. The SMP/E package contains four FMIDs that can be selectively installed based on which parts are needed.

Rational Team Concert for System z prerequisites

The Rational Team Concert for System z client and Jazz Team Server for System z include the software required for basic operation. The Rational Build Agent and the Jazz Build Engine have additional prerequisites.

Supported environments

- Microsoft Windows 2003 Server with Service Pack 2 (32 bit and 64 bit)
- Microsoft Windows 2008 Server (32 bit and 64 bit)
- Microsoft Windows XP (32 and 64bit)
- Red Hat Enterprise Linux 5.0 Update 1
  - x86, 32-bit and 64-bit
  - System z, 64-bit
- SUSE Linux Enterprise Server 10 with Service Pack 1
  - x86, 32-bit and 64-bit
  - System z, 64-bit
- z/OS 1.9, or 1.10 (recommended)

Note: The Rational Team Concert for System z client is not supported on any 64 bit platform.

Additional Required Software

- Apache Derby 10.3.2.2 (included)
- Apache Tomcat 5.5 (included)
- Windows and Linux x86: IBM JRE 1.5 SR9-0 +IZ45776+IZ49644+IZ49635 (included)
- Linux on System z: IBM JRE 1.5 SR9 IBM JRE 1.5 SR9-SSU (included)
- z/OS : IBM 31 bit SDK for z/OS Java 2 Technology Edition V5 at PTF UK42388 (SR9) or later (not included)

Optional Software

- IBM DB2 z/OS 9.1 (not included)

Supported Build Toolkit Environments

- z/OS V1.9, or z/OS V1.10 (recommended)
- DASD requirements: 30000 tracks in 3390 tracks (maximum - this could be significantly reduced depending on which functions on z/OS are installed/used)
- IBM 31 bit SDK for z/OS Java 2 Technology Edition V5

Supported Client environments

The minimum client hardware is:

- Processor: Intel® Pentium® 3 or 4 (minimum); intel Core 2 Duo (recommended)
- Memory: 1 GB minimum
- Display: 1024 x 768 minimum resolution
- Other hardware: 10/100 mbps wired connectivity or 802.11b

The client is supported on the following 32-bit operating systems:

- Windows® Vista with Service Pack 1
- Windows XP with Service Pack 2
- Red Hat Enterprise Linux 5.0 Update 1
- SUSE Linux Enterprise Desktop 10 with Service Pack 1

Additional Required Software (included)

- IBM® JRE 1.5 Service Release SR9-0 +IZ45776+IZ49644+IZ49635

Additional prerequisites for the Rational Build Agent and the Build System Toolkit

The Build System Toolkit has the following additional prerequisites:

Additional permissions

The user ID associated with the Jazz Build Engine requires read, write, and execute permissions to the installation directory and its subdirectories. See the user scenario section for additional directory and data set usage information.

Software requirements

IBM 31-bit SDK for z/OS Java 2 Technology Edition V5 (which includes the required IBM JZOS Batch Launcher) is required and must be installed

and configured separately. For more information, see http://www.ibm.com/servers/eserver/zseries/software/java/j5pcont31.html. The JZOS Java Launcher is required to run the Jazz Build Engine. For more information about the JZOS Batch Launcher, see the *Installation and User's Guide* at http://www.ibm.com/servers/eserver/zseries/software/java/pdf/jzosinstal.pdf.

The Rational Build Agent has the following additional prerequisites:

c89 compiler and UNIX header files
On the z/OS system, the agent runs in the UNIX System Services (USS) environment.

z/OS UNIX shell interface
During installation, you run all commands on z/OS in the z/OS UNIX shell.

Installing the Rational Team Concert for System z Client

About this task

To install RTCz Beta 2, you use the Rational Team Concert for System z launchpad. Although the launchpad has an option for updating a previous installation, this is not supported for Beta 2. If you participated in a previous beta driver, you can not update this installation or migrate any code created during use of the previous beta driver.

1. Start the launchpad application. The launchpad application is in the "rtc" directory created when you unzipped the installation package.
   a. To run the file on Windows, run the command: launchpad.exe.

      Note: You must have administrator privileges to run the launchpad.
   b. To run the file on Linux, run the command: sh launchpad.sh.

      Note: The Linux commands to start the launchpad program are only for the RTCz for Eclipse IDE.
2. Select **New Installation** on the launchpad.
3. Select **Rational Team Concert for System z Client**. The Installation Manager wizard starts to guide you through the installation of the client.

What to do next

The user scenario section includes instructions on how and when to connect the Rational Team Concert for System z client to the Jazz Team Server.

Setting up the Jazz Team Server for System z

This section describes the tasks required for configuring and running the Jazz Team Server.

Installing and Configuring the Jazz Team Server for System z on Windows and Linux

About this task

Complete the following steps to install the Rational Team Concert for System z Jazz Team Server:

1. Select **New Installation** on the launchpad. The launchpad application is in the "rtc" directory created when you unzipped the installation package.
2. Select **Jazz Team Server for System z**. The Installation Manager wizard starts to guide you through the installation of the server. The Jazz Team Server for System z is installed by default to `\Program Files\IBM\JazzTeamServer` on Windows and `/opt/IBM/JazzTeamServer` on Linux. The installation directory is referred to later in this guide as *JazzInstallDir*. If you chose a different path, please use that path for *JazzInstallDir*.

What to do next

You are now ready to start the Apache Tomcat server.

Starting and stopping the Apache Tomcat server

Apache Tomcat is installed in the directory *JazzInstallDir*`/jazz/server/tomcat`. The Web application (jazz.war) has been installed in the Apache Tomcat directory `webapps`. In a command window, set your current directory to *JazzInstallDir*`/jazz/server/`. The server startup and shutdown scripts are located in this directory.

About this task

As part of this beta package, the open source application server Tomcat and the Derby database are also installed. Derby has a 10 user maximum. These supporting applications are installed pre-configured. Accepting the defaults enables you to start the server as soon as the installation is complete. If you want to configure a Websphere Application Server instance to run the Jazz server or use a DB2 database, you can find instructions on how to do this in the online help of the Rational Team Concert for System z client installation.

1. Complete these **optional** steps to start the Apache Tomcat server:
   a. Apache Tomcat is configured to use the ports 9080 and 9443 in file *JazzInstallDir*`/jazz/server/tomcat/conf/server.xml`. If necessary, change the ports according to your system setup. If you change the ports, you also need to update the configuration settings of the Jazz server repository HTTP ports in the **Advanced Properties** configuration panel of the Jazz Administrative Web user interface later during setup.
2. To start the server, from the *JazzInstallDir*`/server` directory, run this command: `server.startup.bat`
3. A separate Apache Tomcat console window opens and several informational messages launch, including a message about the Apache Tomcat Native Library. These informational messages do not affect the Jazz Team Server functionality.

   Important: Do not close the Apache Tomcat console window or the server will stop.
4. The directory *JazzInstallDir*`/jazz/server/tomcat/logs` contains the server log files. If you cannot start the server, check the log files.

What to do next

Tip: When you are ready to stop the server, from the *JazzInstallDir*/server directory, run this command: server.shutdown.bat.

Running the setup wizard

Running the setup wizard verifies that the server is operating properly and guides you through some additional basic steps to configure the server.

Before you begin

To verify that the Jazz Team Server is connecting to the database, look at the server log or console output. The connection and database information is echoed on its first access. The directory *JazzInstallDir*/jazz/server/tomcat/logs is used for the server log files. This procedure assumes your server is available using the hostname localhost and the default port 9443. If necessary, replace localhost with your server hostname and replace the port number.

About this task

Complete the following steps to run the setup wizard:

1. Start the setup wizard to configure your server. Use the URL: https://localhost:9443/jazz/setup.
2. Enter the default user name and password:
   - User name: ADMIN
   - Password: ADMIN

   Note: User name and password are case-sensitive.
3. Choose one of the two setup paths available on the setup wizard:
   - The **Fast Path Setup** uses the default configuration. Use this option if you want to get the server running quickly. Using the fast path, you set up the user registry.
   - The **Custom Setup** guides you through the detailed server configuration. During the custom setup, you set up the following items:
     – Database
     – E-mail notification
     – User registry

   Note: For the Rational Team Concert for System z beta, the Tomcat user registry is the supported user registry and Derby is the supported database. It is sufficient for the purpose of this beta to run the **Fast Path Setup**. Create an administrator, assign a developer license, and then use this ID to get started on the user scenarios provided later in this document. Instructions for creating a project and a team are included in this guide in *Scenario 1: Getting started with Rational Team Concert for System z*.

What to do next

When the initial setup is complete, you can configure additional options using the Jazz Team Server Administrative Web user interface using this URL: https://localhost:9443/jazz/admin.

Adding your server ports to the Jazz Team Server configuration

Port numbers are used for composing URLs for things like feed links and item links in e-mail notifications. If you use port numbers other than the default port numbers, then you must update your Jazz Team Server configuration to use your port numbers.

To update your configuration, use the Advanced Properties configuration page of the administrative Web user interface to modify the following properties:

Property name:
com.ibm.team.repository.servlet.internal.ServletConfigurationService

- Repository HTTP port
- Repository HTTPS port

Property name:
com.ibm.team.repository.service.internal.webuiInitializer.ConfigPropertyInitializer

- Repository HTTP port
- Repository HTTPS port

You are now ready to connect to the server with the Rational Team Concert client or a Web browser. See *Scenario 1: Getting started with Rational Team Concert for System z* later in this guide.

Installing and configuring the Jazz Team Server for System z on z/OS

This information provides step-by-step instructions for configuring the Jazz Team Server for System z® on z/OS®.

These instructions are intended for system administrators responsible for installing and maintaining the Jazz™ Team Server for System z on z/OS. Instructions for installing the Rational® Team Concert client on your workstation are also included. To understand the instructions, you should be familiar with the Windows® and z/OS operating systems.

Software prerequisites

Before you can install Rational Team Concert for System z on z/OS, you must ensure that the prerequisite applications are installed and operational. Be aware of optional software requirements. The Jazz Team Server for System z requires that an application server and a database be configured.

For this beta program you can choose one of the following application server and database options:

- Apache Tomcat 5.5 application server and Apache Derby database (both included with the Jazz Team Server for System z download package)
- Apache Tomcat 5.5 application server (included) and DB2 Version 9.1 for z/OS (not included)

Restriction: WebSphere Application Server is not supported for this beta program, but will be an available option when Rational Team Concert for System z v2.0 is released.

Tip: These instructions will indicate whether specific steps are required or optional depending on which application server and database combination you choose.

These software products are required for installing Rational Team Concert for System z 2.0 Beta 2:

- z/OS V1.9 or z/OS V1.10 (recommended).
- IBM® 31-bit SDK for z/OS Java™ 2 Technology Edition, V5, which includes the IBM JZOS Batch Launcher. The minimum service level required is UK42388(SR 9). The Jazz Team Server for System z depends on IBM Java JRE 1.5 SR 9 or greater.[1]

The following software is optional:

- DB2® Version 9.1 for z/OS, with the Universal JDBC driver configured to ensure interoperability between the database and z/OS[2]

Notes:

1. For more information on the IBM 31-bit SDK for z/OS Java 2 Technology Edition V5, which includes the JZOS Java Launcher and Toolkit, see IBM 31-bit SDK for z/OS Java 2 product pages at http://www.ibm.com/servers/eserver/zseries/software/java/j5pcont31.html. The JZOS Java Launcher is required to run the Jazz Team Server repository tools. For more information about the JZOS Batch Launcher, see the Installation and User's Guide at http://www.ibm.com/servers/eserver/zseries/software/java/pdf/jzosinstal.pdf.
2. See Using the DB2 Universal JDBC Driver to access DB2 for z/OS at http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.ejbfep.multiplatform.doc/info/ae/ae/tdat_jdbcdb2cfg.html.

SMP/E installation

This topic provides information related to installing Rational Team Concert for System z on z/OS. .

The instructions assume you have already completed the Rational Team Concert for System z System Modification Program/Extended (SMP/E) installation. The SMP/E instructions are in the *Program Directory* included with the product download package.

Notes about the Program Directory:

- The *Program Directory* refers to specific path names, if you extract your files to a different location, you must update the path names to match your location.
- In some cases, the *Program Directory* refers to documentation for Rational Team Concert for System z V1.0.1 for additional configuration tasks. Instead use this beta "Installation and User's Guide" for all configuration tasks.

The SMP/E installation installs two MVS™ datasets, *hlq*.SBLZSAMP and *hlq*.SBLZAUTH, where *hlq* is a high-level qualifier chosen during the SMP/E Installation. As noted in the *Program Directory*, *hlq*.SBLZAUTH should be APF authorized. Many of the additional configuration steps in this guide involve editing and submitting JCL that is found in the *hlq*.SBLZSAMP data set.

The SMP/E installation also installs one z/OS UNIX® System Services file directory*/@pathPrefix@*/usr/lpp/jazz.

Note: */@pathPrefix@* is the path prefix specified during the SMP/E installation.

After you complete the SMP/E installation, follow the subsequent instructions for the following tasks:

- Complete the installation and customization on z/OS
- Configure the Jazz Team Server for System z
- Install the Rational Team Concert client on your computer

Configuring your installation

Before you can configure your Jazz Team Server on z/OS, you must ensure that the SMP/E installation has ended successfully.

Creating additional Jazz Team Server directories (required)

The Jazz Team Server on z/OS requires two directories in addition to the USS directory created by the SMP/E installation. One is a directory used to store Jazz Team Server for System z configuration files and the other is a working directory. Sample member `BLZCPDIR` in *`hlq`*`.SBLZSAMP` is used to create and populate these directories with the required files from the SMP/E installed directory.

The following three symbolic references are used throughout this guide to refer to these particular directories:

Symbolic name	Use	Variable in BLZCPDIR	Default directory
@pathPrefix@	The path prefix specified during the SMP/E install.	HOME	
@confPath@	The Jazz Team Server for System z configuration files directory.	CONF	`/etc/jazz`
@workPath@	The Jazz Team Server for System z working directory.	WORK	`/u/jazz`

Space recommendations:
The Jazz Team Server for System z directories in Beta 2 should have the following allocations:

@workPath@
700 cylinders 3390 (assuming you are going to run the Jazz Team Server for System z on z/OS)

@confPath@
2 cylinders 3390 (could be combined with @workPath@)

RACF file access requirements: The Jazz Team Server for System z directories require the following RACF file access permissions:

1. @confPath@ must be accessible for read and write by the user ID under which the Tomcat server executes.
2. @workPath@ must be accessible for read and write by the user ID under which the Tomcat server executes.
3. @workPath@ must be accessible for read and write by the user ID under which the Jazz Team Server for System z repository tools are run.

Configure member `BLZCPDIR` in *`hlq`*`.SBLZSAMP` using the instructions contained in the sample. Use the SUBMIT command to submit the modified JCL and check the job log. If everything is set up properly, all steps end with return code 0.

Customizing the configuration files (required)

Several Jazz Team Server for System z configuration files must be modified based on the directories you have chosen for `@pathPrefix@`, `@workPath@`, and `@confPath@`. The topics in this section provide instructions for modifying the configuration files.

The Jazz Team Server for z/OS properties files are ASCII files. You can use one of the following techniques to edit ASCII files under z/OS UNIX:

- Download or use FTP to transfer the files to a Windows PC, modify them, and then transfer them back to the z/OS system.
- If you use Rational Developer for System z, use Remote System Explorer to connect and modify the files.
- If you have z/OS 1.10 or z/OS 1.9 with the PTF for APAR OA22250, use ISPF option 3.17 to edit ASCII files under USS.
- If you have other tools for editing ASCII files under USS, use those tools.

Customizing the provisioning profiles (required):

This topic describes what you must change in the provisioning profiles in order to customize them.

About this task

If you have used the `@pathPrefix@`, you should edit the following files:

`@confPath@/provision_profiles/profile.ini`
Change `url=file:/usr/lpp/jazz/server/update-site` to `url=file:/@pathPrefix@/usr/lpp/jazz/server/update-site`.

For example, if `@pathPrefix@` is set to `myroot`, then this setting resolves to `url=file:/myroot/usr/lpp/jazz/server/update-site`.

`@confPath@/provision_profiles/rtcz-license-profile.ini`
Change `url=file:/usr/lpp/jazz/server/license-update-site` to `url=file:/@pathPrefix@/usr/lpp/jazz/server/license-update-site`.

For example, if `@pathPrefix@` is set to `myroot`, then this setting resolves to `url=file:/myroot/usr/lpp/jazz/server/license-update-site`.

`@confPath@/provision_profiles/nl1profile.ini`
Change `url=file:/usr/lpp/jazz/server/nl1-update-site` to `url=file:/@pathPrefix@/usr/lpp/jazz/server/nl1-update-site`.

For example, if `@pathPrefix@` is set to `myroot`, then this setting resolves to `url=file:/myroot/usr/lpp/jazz/server/nl1-update-site`.

`@confPath@/provision_profiles/nl2profile.ini`
Change `url=file:/usr/lpp/jazz/server/nl2-update-site` to `url=file:/@pathPrefix@/usr/lpp/jazz/server/nl2-update-site`.

For example, if `@pathPrefix@` is set to `myroot`, then this setting resolves to `url=file:/myroot/usr/lpp/jazz/server/nl2-update-site`.

`@confPath@/provision_profiles/rtcz-profile.ini`
Change `url=file:/usr/lpp/jazz/server/update-site` to `url=file:/@pathPrefix@/usr/lpp/jazz/server/update-site`.

For example, if `@pathPrefix@` is set to `myroot`, then this setting resolves to `url=file:/myroot/usr/lpp/jazz/server/update-site`.

Customizing the logging utility files:

The @workPath@/catalina_base/log4j.properties file configures the logging framework used by the Jazz Team Server for System z.

By default, the log4j.properties file writes Jazz Team Server messages to the Tomcat Application Server stdout file (generally, the application server job log), as well as to a separate file that is specified in the log4j.properties file. Edit the @workPath@/catalina_base/log4j.properties file and locate the line:

```
log4j.appender.file.File=@workPath@/logs/jazz.log
```

This line should be updated to reflect the value that you are using for your Jazz Team Server for System z@workPath@ directory, for example:

```
log4j.appender.file.File=/u/jazz/logs/jazz.log
```

Review Tomcat port configuration (recommended)

To avoid port conflicts, review the Tomcat application server default port settings to make sure they are not the same as other application server instances that you have installed.

About this task

By default, the Tomcat application server included with this beta package uses port 9080 for a nonsecure port and port 9443 for a secure port (as well as ports 9005, 9009, and 8082). These default ports might conflict with a default WebSphere Application Server instance that you have installed. Review the Tomcat server.xml file, which defines these ports and is located at @workPath@/catalina_base/conf/server.xml.

Tip: If you modify the default ports in the server.xml file, record the changes so that you can substitute these ports in when you complete the scenario tasks.

Choose one of the following options based on the application server and database you are using:

- Running the Jazz Team Server for System z with Tomcat and Derby
- Running the Jazz Team Server for System z with Tomcat and DB2 for z/OS

Running the Jazz Team Server for System z with Tomcat and Derby

Running the Jazz Team Server for System z with Tomcat and Derby involves several sample JCL members and configuration property files.

Before you begin

These instructions assume that you are using the sample teamserver_derby.properties file. If you use a different properties file name, change the sample JCL members to point to your properties file.

1. Edit the file **teamserver_derby.properties** in the @confPath@ directory.
2. Change com.ibm.team.repository.db.jdbc.location = **@workPath@**/repositoryDB to your path, for example, com.ibm.team.repository.db.jdbc.location = **/u/jazz**/repositoryDB
3. Also change **@workPath@** to your path in the following properties:
   a. com.ibm.team.fulltext.indexLocation=**@workPath@/workitemindex**
   b. com.ibm.team.repository.tmpdir=**@workPath@/contentservice**
   c. com.ibm.team.scm.tmpdir=**@workPath@/contentservice**
   d. com.ibm.team.scm.vcs.tmpdir=**@workPath@/versionedcontentservice**

Starting Tomcat using the Derby database: At this point you are ready to start Tomcat using the default Derby database. If you need to run other Jazz Team Server for System z repository tools functions, like importing or exporting data for migration or recreating a new database, see Appendix D, "Additional Jazz Team Server for System z repository tools tasks on z/OS," on page 89.

Configure member `BLZSRV2` in *`hlq`*`.SBLZSAMP` using the instructions contained in the sample. Use the SUBMIT command to submit the modified JCL and check the job log carefully for errors and messages that the server startup has completed. You can adjust the WLM Service Class for the `BLZSRV2` job to meet your system requirements.

You can stop the Jazz Team Server for System z by stopping this job.

Completing the installation:

The tasks required to complete the installation should be performed by your Jazz Team Server for System z administrator. After the server is installed, you must consider some configuration options before continuing.

About this task

Some files contain passwords. Those files should be protected so that they are readable only by users who are authorized to know the account passwords.

-
  - **teamserver.properties** - The Jazz Team Server requires that the database password is stored in `JazzInstallDir/server/conf/jazz/teamserver.properties`.

    When properties files are saved, the application always makes a backup copy of the previous version in the same directory. If you want to remove all files that contain the clear-text password, remove the backup properties files after configuring the server for the first time.
- When connecting to the server with the Rational Team Concert for System z client or a Web browser, you might see security certificate warnings. To disable the warning, see Disabling security certificate settings at http://publib.boulder.ibm.com/infocenter/rtc/v1r0m0/index.jsp?topic=/com.ibm.team.install.doc/topics/t_disable_server_certificates.html.
- You are now ready to connect to the server with the Rational Team Concert client or a Web browser.

Running the Jazz Team Server for System z with Tomcat and DB2 for z/OS

Before you begin

Running the Jazz Team Server for System z with Tomcat and DB2 for z/OS involves several sample JCL members and configuration property files. These instructions assume that you are using the sample `teamserver_db2z.properties` file. If you use a different properties file name, change the sample JCL members to point to your properties file.

About this task

Using a database with DB2 on z/OS requires additional configuration steps.

Setting up a DB2 database:

The DB2 setup tasks should be performed by your DB2 administrator.

DB2 prerequisites:

Ensure that DB2 Version 9.1 for z/OS with the Universal JDBC driver is installed and running on the z/OS system that is used as the database server. This might not be the same system on which the Jazz Team Server runs.

The Jazz Team Server for System z requires the WLM procedures associated with the DB2 stored procedures SYSPROC.DSNUTILS and SYSIBM.SQLxxx to be active. If necessary, verify the stored procedures are active by comparing the names of the DB2 WLM environment variables with the active WLM procedures. Use SQL SELECT command to retrieve the WLM procedure names through DB2 SPUFI or your preferred technique:

```
SELECT DISTINCT WLM_ENVIRONMENT FROM SYSIBM.SYSROUTINES WHERE
  (NAME='DSNUTILS' OR (SCHEMA='SYSIBM' AND NAME LIKE 'SQL%'));
```

This command produces results like the following:

```
---------+---------+---------+---------+---------+---------+---
WLM_ENVIRONMENT
---------+---------+---------+---------+---------+---------+---
DSN9WLM1
```

Use the following command from the z/OS console to display the WLM active procedures: D WLM,APPLENV=*. This command produces results like the following:

```
SDSF SYSLOG   2493.101 2094 2094 07/10/2008 0W   14886  COMMAND ISSUED
 RESPONSE=RALNS32
  IWM029I  16.31.12  WLM DISPLAY 465
    APPLICATION ENVIRONMENT NAME     STATE     STATE DATA
    BBOASR1                          AVAILABLE
    BBOASR2                          AVAILABLE
    BBOC001                          AVAILABLE
    CBINTFRP                         AVAILABLE
    CBNAMING                         AVAILABLE
    CBSYSMGT                         AVAILABLE
    DSN9WLM1                         AVAILABLE
    DSN9WLM2                         AVAILABLE
    DSN9WLM3                         AVAILABLE
```

In this case, you can see that the DSN9WLM1 procedure is active.

Setting up DB2 for z/OS to use with the Jazz Team Server for System z:

When running the Jazz Team Server for System z with DB2 for z/OS, you must create a DB2 storage group and a DB2 database. You must also authorize the Jazz Team Server for System z user to that storage group and database.

About this task

The following steps must be performed before running the repository tools database builder utility, which creates the Jazz repository tables in your database instance. None of these steps is performed by the Jazz Team Server database builder utility.

Create a storage group

The storage group must be appropriate to the machine. The following example shows a DB2 SQL create statement:

```
CREATE STOGROUP JAZZSTG VOLUMES ('*') VCAT yourHlq ;
```

Notes:

1. The storage group can be named something other than *JAZZSTG*.
2. The storage group name is used later for the Jazz Team Server for System z **teamserver_db2z.properties** **com.ibm.team.repository.db.db2.dsn.stogroup**property.
3. *yourHlq* is the high level qualifier of your DB2 files. It must exist on your system, and the Jazz Team Server for System z user must have full access to it.

Create a database

The database must be appropriate to the system. The following example shows a DB2 SQL create statement:

```
CREATE DATABASE JAZZDB STOGROUP JAZZSTG BUFFERPOOL bp8kx ;
```

Notes:

1. The database can be named something other than *JAZZDB*.
2. The database name is used later for the Jazz Team Server for System z**teamserver_db2z.properties** **com.ibm.team.repository.db.db2.dsn.dbname** property.
3. *bp8kx* is the buffer pool name, for example **BP8K0** (The Jazz Team Server for System z requires an 8K page size)
4. The buffer pool name is used later for the Jazz Team Server for System z **teamserver_db2z.properties** **com.ibm.team.repository.db.db2.dsn.bufferpool** properties.
5. You can define multiple Jazz DB2 databases in a DB2 subsystem to contain separate Jazz repositories. This is done in conjunction with the Jazz Server `teamserver_db2z.properties` file directive `com.ibm.team.repository.db.schemaPrefix`, as discussed in "Customizing Jazz Team Server for System z properties files for DB2 for z/OS."

Authorize a Jazz Team Server for System z user

The Jazz Team Server for System z requires a user ID and password to access the Jazz Team Server for System z DB2 repository. The user ID and password are specified later in the **teamserver_db2z.properties** file. This user ID is not used to log on to the Jazz Team Server for System z, it is used only to provide authority for the Jazz Team Server for System z to access the DB2 for z/OS database. Specifically, this user ID requires permissions as shown here. In the following example, the user has the name *jazz*.

```
GRANT DBADM ON DATABASE jazzdb TO jazz ;
GRANT USE OF STOGROUP jazzstg TO jazz ;
GRANT USE OF BUFFERPOOL bpx TO jazz ;
COMMIT ;
```

In addition, if the value of field DBADM CREATE AUTH is set to NO on panel **DSNTIPP** during DB2 installation, you must grant SYSADM authorization to the Jazz Team Server for System z user.

```
GRANT SYSADM TO jazz ;
COMMIT ;
```

Customizing Jazz Team Server for System z properties files for DB2 for z/OS:

This section describes how to edit the `teamserver_db2z.properties` file to configure the database connection.

About this task

Use the DIS DDF command for your DB2 for z/OSsubsystem to display some of the values you need to supply. For example, you can retrieve the *location*, *ipaddr*, and *port* (*tcpport*) from the following display:

```
-DSN9 DIS DDF
 DSNL080I  -DSN9 DSNLTDDF DISPLAY DDF REPORT FOLLOWS: 548
 DSNL081I STATUS=STARTD
 DSNL082I LOCATION           LUNAME            GENERICLU
 DSNL083I NS32DB             NETA.NS32DB       -NONE
 DSNL084I TCPPORT=3500  SECPORT=3510  RESPORT=3501  IPNAME=-NONE
 DSNL085I IPADDR=9.42.81.74
 DSNL086I SQL    DOMAIN=RALNS32.rtp.raleigh.ibm.com
 DSNL086I RESYNC DOMAIN=RALNS32.rtp.raleigh.ibm.com
 DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

To configure the server, edit the *@confPath@*/teamserver_db2z.properties file and locate the following lines:

```
com.ibm.team.repository.db.vendor = db2z
com.ibm.team.repository.db.jdbc.location=//ipAddress:ipPort
   /location:user=jazzDBuser;password={password};
com.ibm.team.repository.db.jdbc.password=jazzDBpswd
com.ibm.team.repository.db.db2.dsn.dbname=JAZZDB
com.ibm.team.repository.db.db2.dsn.bufferpool=bp8kx
com.ibm.team.repository.db.db2.dsn.stogroup=JAZZSTG
#com.ibm.team.repository.db.schemaPrefix=xx
```

Edit these lines to match the database configuration you created in previous steps, and also to match your DB2 configuration. Edit the location, user, password, dbname, bufferpool, and stogroup properties according to your configuration.

Specifically, edit the following:

1. In the following line:

   ```
   com.ibm.team.repository.db.jdbc.location=//ipAddress:ipPort/
   location:user=jazzDBuser;password={password};
   ```

   replace:
   - *ipAddress* with your ipaddr.
   - *ipPort* with your tcpport.
   - *location* with the value listed in the DDF report under LOCATION.
   - *jazzDBuser* with the user ID you created, which has appropriate access to the DB2 database.

     Note: Do not modify *password*={password}.
2. In the following line:

   ```
   com.ibm.team.repository.db.jdbc.password=jazzDBpswd
   ```

   replace *jazzDBpswd* with the password for your DB2 user.
3. In the following line:

   ```
   com.ibm.team.repository.db.db2.dsn.dbname=JAZZDB
   ```

   replace *JAZZDB* with the name of the database you created.
4. In the following line:

   ```
   com.ibm.team.repository.db.db2.dsn.bufferpool=bp8kx
   ```

   replace *bp8kx* with the bufferpool you chose when you created the database.
5. In the following line:

```
com.ibm.team.repository.db.db2.dsn.stogroup=JAZZSTG
```

replace *JAZZSTG* with the storage group you created.

6. In the following line:

```
#com.ibm.team.repository.db.schemaPrefix=xx
```

you can remove the # and replace *xx* with a prefix of your choice. This is optional. It enables you to have multiple Jazz repositories in one DB2 subsystem.

Note: To create several Jazz databases in the same DB2 subsystem, you must differentiate the table owners for the Jazz tables. To do so, the Jazz Team Server for System z uses the `com.ibm.team.repository.db.schemaPrefix` directive to add a prefix to the Jazz DB2 objects so that they are unique within a DB2 subsystem. The prefix set in `com.ibm.team.repository.db.schemaPrefix` will be added to the owner along with an underscore. For example, table WORKITEMS_SNAPSHOT.WORKITEM_CHNGS will be created as X1_WORKITEMS_SNAPSHOT.WORKITEM_CHNGS, in a given database when `com.ibm.team.repository.db.schemaPrefix=`**`X1`**.

In addition, replace all instances of *@workPath@* with the path that you selected for *@workPath@* in the following properties:

- `com.ibm.team.fulltext.indexLocation=@workPath@/workitemindex`
- `com.ibm.team.repository.tmpdir=@workPath@/contentservice`
- `com.ibm.team.scm.tmpdir=@workPath@/contentservice`
- `com.ibm.team.scm.vcs.tmpdir=@workPath@/versionedcontentservice`

Note: Jazz Team Server for System z will create these directories.

Creating database tables using Jazz Team Server for System z repository tools:

When you use DB2 for z/OS, initially you must use the Jazz Team Server for System z repository tools to create the required database tables and indexes for the Jazz Team Server for System z.

About this task

Ensure the `teamserver_db2z.properties` file has been configured correctly as described in "Customizing Jazz Team Server for System z properties files for DB2 for z/OS" on page 13. Then you must:

1. Configure member `BLZCREDB` in *`hlq`*`.SBLZSAMP` by following the instructions in the sample JCL.
   - The Jazz Team Server for System z and the Jazz Team Server repository tools require access to the DB2 V9 JDBC license jar file on your system. This file is named `db2jcc_license_cisuz.jar` and is located by default at `/usr/lpp/db2910_jdbc/classes`.
   - If the file is in a different directory, then modify the sample job to point to the correct location.
2. Submit the modified JCL and check the job log. The following message must end the STDOUT: `The database tables were created successfully.` The details are in the JCL sample.

Starting Tomcat using the DB2 for z/OS database: At this point you are ready to start Tomcat using the DB2 for z/OS database. If you need to run other Jazz Team

Server for System z repository tools functions, like importing or exporting data for migration or recreating a new database, see Appendix D, “Additional Jazz Team Server for System z repository tools tasks on z/OS,” on page 89.

Configure member `BLZSRV1` in *`hlq`*`.SBLZSAMP` using the instructions contained in the sample. Use the SUBMIT command to submit the modified JCL and check the job log carefully for errors and messages that the server startup has completed. You can adjust the WLM Service Class for the `BLZSRV1` job to meet your system requirements.

You can stop the Jazz Team Server for System z by stopping this job.

Completing the installation:

The tasks required to complete the installation should be performed by your Jazz Team Server for System z administrator. After the server is installed, you must consider some configuration options before continuing.

About this task

Some files contain passwords. Those files should be protected so that they are readable only by users who are authorized to know the account passwords.

-
  - **teamserver.properties** - The Jazz Team Server requires that the database password is stored in `JazzInstallDir/server/conf/jazz/teamserver.properties`.

    When properties files are saved, the application always makes a backup copy of the previous version in the same directory. If you want to remove all files that contain the clear-text password, remove the backup properties files after configuring the server for the first time.
- When connecting to the server with the Rational Team Concert for System z client or a Web browser, you might see security certificate warnings. To disable the warning, see Disabling security certificate settings at http://publib.boulder.ibm.com/infocenter/rtc/v1r0m0/index.jsp?topic=/com.ibm.team.install.doc/topics/t_disable_server_certificates.html.
- You are now ready to connect to the server with the Rational Team Concert client or a Web browser.

Installing Rational Team Concert for System z on Linux for System z

This information provides step-by-step instructions for installing and configuring Rational Team Concert for System z on Linux® for System z.

These instructions are intended for system administrators responsible for installing and maintaining the Jazz Team Server for System z on Linux for System z. Instructions for installing the Rational Team Concert client on your workstation are also included. To understand the instructions, you must be familiar with Windows and Linux for System z operating systems.

Rational Team Concert for System z builds on the core Rational Team Concert product by providing a Jazz Team Server that runs on Linux for System z.

Introduction

Rational Team Concert for System z V2.0 Beta 2 is supported on Linux for System z. The following limitations apply to installing the product on Linux for System z for this beta program:

- Only the Jazz Team Server for System z is supported on Linux for System z. The Rational Team Concert for System z client and Build System Toolkit for System z are not supported.
- Installing the Jazz Team Server for System z on Linux for System z is completed using Installation Manager. The installation and configuration processes are very similar to the processes used for x86-based Linux, and documentation of those processes generally applies to Jazz Team Server for System z on Linux for System z.
- This document describes the configuration of the Tomcat application server with a Derby database or DB2 for z/OS.
- Use a graphical desktop with your preferred Window Manager to install Rational Team Concert for System z for Beta 2.
- The included *Installation Guide* referenced from the launchpad is not updated with specific Rational Team Concert for System z information. It applies to the base Rational Team Concert product.

Installation prerequisites

Before you can install Rational Team Concert for System z on Linux for System z, you must ensure that the required disk space is allocated and the prerequisite applications are installed and operational.

Rational Team Concert for System z requires the following disk space for installation and temporary files:

- Rational Team Concert for System z installation distribution - 1200 MB (this file can be deleted after it is unpacked)
- Rational Team Concert for System z installation directories - 1200 MB
- Rational Team Concert for System z installed directories (including Installation Manager) - 600 MB

The installation package includes:

- Java Runtime Environment (JRE) for Linux for System z (IBM J9 VM 1.5.0 64-bit).
- Apache Tomcat Version 5.5 Web application server containing the Jazz Team Server Web application.
- A Derby database and all necessary database libraries.

  Important: The Jazz Team Server with a Derby database supports up to 10 users. To support more than 10 users, use DB2.

Make sure you have the following prerequisite applications installed and operational before you proceed with the installation on Linux for System z:

- **Operating system:** RedHat Enterprise Linux, Version 5 Update 1 or later, or SUSE Linux Enterprise Server, Version 10 Service Pack 1.
- **Application Server:** Apache Tomcat Server V5.5 (included)
- **Optional**: DB2 V9.1 for z/OS with the Universal JDBC driver configured to ensure interoperability between the database and z/OS.

Notes:

- The Derby database included in the default configuration requires no installation, but this database has a 10 user limit.
- DB2 for z/OS is obtained separately. These instructions assume that you have completed the basic installation configuration steps for DB2 for z/OS. See Using the DB2 Universal JDBC Driver to access DB2 for z/OS.

Overview of installing Jazz Team Server for System z on Linux for System z

You can install the Jazz Team Server for System z Beta 2 using Installation Manager.

Before you begin

For best results working with the Installation Manager, establish a graphical desktop session with your Linux for System z system. These instructions do not include details about configuring your Linux on System z system. Some of the options include:

SLES 10:

1. Start a VNC server.
2. Connect a VNC viewer to that VNC server.
3. From the VNC viewer, launch a graphical desktop using the kde command.

RedHat:

1. Start a default VNC server.
2. Connect a VNC viewer to that VNC server.
3. From the VNC viewer, launch a graphical desktop using the kde command.

RedHat:

1. Configure the .vnc/xstartup file for the user that will launch the VNC server to support a graphical desktop from VNC.
2. Start this VNC server and connect to it with a VNC viewer.

Configure your database and your server according to your installation environment:

- Apache Tomcat Server with the default Derby database.
- Apache Tomcat Server with a DB2 database on z/OS.

About this task

These instructions provide an overview of the steps required for installing the Jazz Team Server for System z.

1. Ensure you meet all installation and configuration prerequisites.
2. Download and copy the Standard edition of the Jazz Team Server for System z compressed (.zip) file. Extract the contents of the .zip file and use the launchpad script to launch Installation Manager to install the Jazz Team Server for System z.
3. If you plan to use a DB2 database, set up the database to work with Jazz Team Server for System z and create the Jazz Team Server database tables.
4. Verify and complete your installation.

Preparing to install the Jazz Team Server for System z

Before installing the Jazz Team Server for System z, you must complete special conditions for a successful setup on Linux for System z.

- If Security-Enhanced Linux (SELinux) is enabled, you must disable it or change the security context of the Java Runtime Environment (JRE) to allow text relocation in order to install and run Rational Team Concert for System z.

  Note: For more details, see "Configuring Security-Enhanced Linux" on page 27.
- Increase the maximum number of files that the Tomcat user can handle to 5,000. You can do this by adding the following lines to **/etc/security/limits.conf**:

  ```
  tomcat_user hard nofile 5000
  tomcat_user soft nofile 5000
  ```

  Note: tomcat_user should be substituted with the name of the user that launches the Tomcat server.
- The reports component requires that 32-bit X11 libraries be installed on the server. The required packages are **libXp**, **libXinerama**, and **mesa-libGL**.
- Ensure that the user ID that runs the Tomcat server has write access to the **/tmp** directory.

Installing Jazz Team Server for System z on Linux for System z

Download the installation files for Linux for System z from the Jazz download page at http://jazz.net/downloads/rational-team-concert/betas/z2.0Beta2, and extract the contents to a directory on your file system on Linux for System z using the unzip utility. The unzip utility will create a directory called **rtc** under the destination directory for the extracted files.

About this task

The Jazz Team Server for System z installation path names must not contain spaces. A Derby database is included in the default configuration and requires no installation; however, the Derby database has a 10 user limit.

1. From your graphical desk top, under the directory where you extracted the Rational Team Concert for System z installation files, navigate to the `rtc` directory and run launchpad.sh.
2. For Beta 2, the only supported launchpad option is to complete a new installation. Click the link for a new installation.
3. You will be prompted to install both Installation Manager and Jazz Team Server for System z.
4. Follow the Installation Manager prompts to complete the installation.

   Important: Be sure to leave the check box selected to install the optional Tomcat Application Server feature.

   By default, the Jazz Team Server for System z will be installed to the `/opt/ibm/JazzTeamServer` directory. This directory will be referred to as `JazzInstallDir` for the rest of these instructions. If you choose a different directory, substitute your installation directory for the `JazzInstallDir`.

Setting up the database

After your Jazz Team Server for System z is installed, you must configure the database.

Fast path: If you are using the included Derby database, no additional database setup is required. Proceed to "Starting the server" on page 24.

The overview steps for setting up a DB2 database on z/OS are:

1. Create a database.
2. Configure the properties files.
3. Install the DB2 z/OS license files on your Linux on System z system.
4. Create the database tables.

If you are using DB2, proceed to "Setting up a DB2 database on DB2 for z/OS."

Setting up a DB2 database on DB2 for z/OS:

Set up a DB2 database to work with the Jazz Team Server for System z.

Before you begin

If you are using a DB2 database on z/OS, the Jazz Team Server for System z and the Jazz Team Server for System z repository tools need access to the JDBC license .jar file that is shipped with DB2 for z/OS. To set access to the license file, complete the following steps:

- Create a directory called `db2z` under the `JazzInstallDir/server` directory.
- In the directory you created, place a copy of the `db2jcc_license_cisuz.jar` file from your z/OS system. You can find this .jar file in the DB2 9.1 SDSNJCC file system, which is typically mounted at a file path like `/usr/lpp/db2910_jdbc.` The `db2jcc_license_cisuz.jar` file is located in the `classes` subdirectory of that directory.

Setting up a DB2 database

The DB2 setup tasks should be performed by your DB2 administrator.

DB2 prerequisites

Ensure that DB2 Version 9.1 for z/OS with the Universal JDBC driver is installed and running on the z/OS system that is used as the database server.

The Jazz Team Server for System z requires the WLM procedures associated with the DB2 stored procedures SYSPROC.DSNUTILS and SYSIBM.SQLxxx to be active. If necessary, verify the stored procedures are active by comparing the names of the DB2 WLM environment variables with the active WLM procedures. Use SQL SELECT command to retrieve the WLM procedure names through DB2 SPUFI or your preferred technique:

```
SELECT DISTINCT WLM_ENVIRONMENT FROM SYSIBM.SYSROUTINES WHERE
  (NAME='DSNUTILS' OR (SCHEMA='SYSIBM' AND NAME LIKE 'SQL%'));
```

This command produces results like the following:

```
---------+---------+---------+---------+---------+---------+---
WLM_ENVIRONMENT
---------+---------+---------+---------+---------+---------+---
DSN9WLM1
```

Use the following command from the z/OS console to display the WLM active procedures: D WLM,APPLENV=*. This command produces results like the following:

```
SDSF SYSLOG   2493.101 2094 2094 07/10/2008 0W   14886  COMMAND ISSUED
 RESPONSE=RALNS32
  IWM029I  16.31.12  WLM DISPLAY 465
    APPLICATION ENVIRONMENT NAME     STATE     STATE DATA
    BBOASR1                          AVAILABLE
```

```
BBOASR2                          AVAILABLE
BBOC001                          AVAILABLE
CBINTFRP                         AVAILABLE
CBNAMING                         AVAILABLE
CBSYSMGT                         AVAILABLE
DSN9WLM1                         AVAILABLE
DSN9WLM2                         AVAILABLE
DSN9WLM3                         AVAILABLE
```

In this case, you can see that the DSN9WLM1 procedure is active.

Setting up DB2 for z/OS to use with the Jazz Team Server for System z

When running the Jazz Team Server for System z with DB2 for z/OS, you must create a DB2 storage group and a DB2 database. You must also authorize the Jazz Team Server for System z user to that storage group and database.

About this task

The following steps must be performed before running the repository tools database builder utility, which creates the Jazz repository tables in your database instance. None of these steps is performed by the Jazz Team Server database builder utility.

Create a storage group

The storage group must be appropriate to the machine. The following example shows a DB2 SQL create statement:

```
CREATE STOGROUP JAZZSTG VOLUMES ('*') VCAT yourHlq ;
```

Notes:

1. The storage group can be named something other than *JAZZSTG*.
2. The storage group name is used later for the Jazz Team Server for System z **teamserver.properties com.ibm.team.repository.db.db2.dsn.stogroup**property.
3. *yourHlq* is the high level qualifier of your DB2 files. It must exist on your system, and the Jazz Team Server for System z user must have full access to it.

Create a database

The database must be appropriate to the system. The following example shows a DB2 SQL create statement:

```
CREATE DATABASE JAZZDB STOGROUP JAZZSTG BUFFERPOOL bp8kx ;
```

Notes:

1. The database can be named something other than *JAZZDB*.
2. The database name is used later for the Jazz Team Server for System z**teamserver.properties com.ibm.team.repository.db.db2.dsn.dbname** property.
3. *bp8kx* is the buffer pool name, for example **BP8K0** (The Jazz Team Server for System z requires an 8K page size)
4. The buffer pool name is used later for the Jazz Team Server for System z **teamserver.properties com.ibm.team.repository.db.db2.dsn.bufferpool** properties.
5. You can define multiple Jazz DB2 databases in a DB2 subsystem to contain separate Jazz repositories. This is done in conjunction with the Jazz Server `teamserver.properties` file directive

com.ibm.team.repository.db.schemaPrefix, as discussed in "Customizing Jazz Team Server for System z properties files for DB2 for z/OS" on page 13.

Authorize a Jazz Team Server for System z user

The Jazz Team Server for System z requires a user ID and password to access the Jazz Team Server for System z DB2 repository. The user ID and password are specified later in the **teamserver.properties** file. This user ID is not used to log on to the Jazz Team Server for System z, it is used only to provide authority for the Jazz Team Server for System z to access the DB2 for z/OS database. Specifically, this user ID requires permissions as shown here. In the following example, the user has the name *jazz*.

```
GRANT DBADM ON DATABASE jazzdb TO jazz ;
GRANT USE OF STOGROUP jazzstg TO jazz ;
GRANT USE OF BUFFERPOOL bpx TO jazz ;
COMMIT ;
```

In addition, if the value of field DBADM CREATE AUTH is set to NO on panel **DSNTIPP** during DB2 installation, you must grant SYSADM authorization to the Jazz Team Server for System z user.

```
GRANT SYSADM TO jazz ;
COMMIT ;
```

Customizing Jazz Team Server for System z properties files for DB2 for z/OS

Jazz Team Server for System z uses the configuration file located in JazzInstallDir/server/conf/jazz by default for configuration information about the database connection.

Edit the teamserver.properties file. Change the following lines to comments so that the default Derby database will not be used. Specifically, change the following lines:

```
# JDBC DB location, specifying this property disables system-based selection
# of default location
com.ibm.team.repository.db.vendor = DERBY
com.ibm.team.repository.db.jdbc.location = repositoryDB
```

to match these lines:

```
# JDBC DB location, specifying this property disables system-based selection
# of default location
#com.ibm.team.repository.db.vendor = DERBY
#com.ibm.team.repository.db.jdbc.location = repositoryDB
```

Uncomment the following properties that start with #com.ibm by removing the #. Then follow the instructions later in this section to set the property values.

```
#
# DB2z configuration
#

# Comment out above lines, uncomment the following lines and customize
# example location to use DB2z
#com.ibm.team.repository.db.vendor = db2z

# Location name of the DB2z sub-system.
# ipAddress, ipPort and location be obtained by running the DB2 command -DISPLAY DDF
#com.ibm.team.repository.db.jdbc.location=//ipAddress:ipPort/
#location:fullyMaterializeLobData=false;user=jazzDBuser;password={password};
#com.ibm.team.repository.db.jdbc.password=jazzDBpswd
```

```
# DB2z Database name
#com.ibm.team.repository.db.db2.dsn.dbname=JAZZDB

# DB2z Bufferpool name
#com.ibm.team.repository.db.db2.dsn.bufferpool=bpx

# DB2z Storage Group name
#com.ibm.team.repository.db.db2.dsn.stogroup=JAZZSTG
```

Use the DIS DDF command for your DB2 for z/OS subsystem to display some of the values you need to supply. For example, you can retrieve the location, IP address (ipaddr), and port (tcpport) from the following display:

```
-DSN9 DIS DDF
DSNL080I -DSN9 DSNLTDDF DISPLAY DDF REPORT FOLLOWS: 548
DSNL081I STATUS=STARTD
DSNL082I LOCATION LUNAME GENERICLU
DSNL083I NS32DB NETA.NS32DB -NONE
DSNL084I TCPPORT=3500 SECPORT=3510 RESPORT=3501 IPNAME=-NONE
DSNL085I IPADDR=9.42.81.74
DSNL086I SQL DOMAIN=RALNS32.rtp.raleigh.ibm.com
DSNL086I RESYNC DOMAIN=RALNS32.rtp.raleigh.ibm.com
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

Edit the following lines to match the database configuration you created in previous steps, and also to match your DB2 configuration. Edit the location, user, password, dbname, bufferpool, and stogroup properties according to your configuration. Specifically, edit the following:

1. In line:

   ```
   com.ibm.team.repository.db.jdbc.location=//ipAddress:ipPort/
   location:fullyMaterializeLobData=false;user=jazzDBuser;password={password};
   ```

   delete:

   ```
   fullyMaterializeLobData=false;
   ```

   Also, replace:

   - `ipAddress` with your IP address.
   - `ipPort` with your TCP port.
   - `location` with the value listed in the DDF report under `LOCATION`.
   - `jazzDBuser` with the user ID you created that has appropriate access to the DB2 database.

   Important: Do not modify `password={password}`.
2. In line:

   ```
   com.ibm.team.repository.db.jdbc.password=jazzDBpswd
   ```

   replace `jazzDBpswd` with the password for your DB2 user.
3. In line:

   ```
   com.ibm.team.repository.db.db2.dsn.dbname=JAZZDB
   ```

   replace `JAZZDB` with the name of the database you created.
4. In line:

   ```
   com.ibm.team.repository.db.db2.dsn.bufferpool=bpx
   ```

replace bpx with the bufferpool you chose when you created the database. Make sure it supports 8K pages. Use a bufferpool such as BP8K0.

5. In line:

   ```
   com.ibm.team.repository.db.db2.dsn.stogroup=JAZZSTG
   ```

   replace JAZZSTG with the storage group you created.

6. Insert the following line if you choose:

   ```
   com.ibm.team.repository.db.schemaPrefix=xx
   ```

   replace xx with a prefix of your choice. This is optional. It enables you to have multiple Jazz repositories in one DB2 subsystem.

   Note: If you want to create several Jazz databases in the same DB2 subsystem, you must differentiate the table owners for the Jazz tables. In order to do that, the Jazz Team Server for System z uses the com.ibm.team.repository.db.schemaPrefix directive to add a prefix to the Jazz DB2 objects so that they are unique within a DB2 subsystem. The owner will include the prefix set in com.ibm.team.repository.db.schemaPrefix along with an underscore. For example, the table WORKITEMS_SNAPSHOT.WORKITEM_CHNGS will be created as X1_WORKITEMS_SNAPSHOT.WORKITEM_CHNGS, in a given database when com.ibm.team.repository.db.schemaPrefix=**X1**.

Creating the Jazz Team Server database tables

Create the database tables using the repository tools.

About this task

To create the database tables:

Run the command **repotools.sh -createTables** to create the database tables for a Jazz repository.
The **repotools.sh** script is located in the JazzInstallDir/server directory.
The command uses the configuration properties in **teamserver.properties** for the connection and size settings. By default, the command looks in the current directory.

- Execute the following command:

  ```
  ./repotools.sh -createTables
  ```

This command creates the tablespace and all the required tables and indexes for a Jazz Team Server repository.

Starting the server

This topic describes the different options for running the server startup scripts.

Starting the Apache Tomcat server:

This topic describes how to start the Apache Tomcat Server.

Before you begin

Apache Tomcat has been installed in the directory JazzInstallDir/server/tomcat. The Web application (jazz.war) has been installed in the Apache Tomcat directory webapps. In a command window, set your current directory to JazzInstallDir/server. The server startup and shutdown scripts are located in this directory.

- If you want to run the start and stop scripts from any other directory, you must change the `.ini` files in `JazzInstallDir/server/conf/jazz/provision_profiles` to use an absolute path.
- You can also use the **Start Jazz Team Server** option from your main application start tab in the graphical desktop in the Jazz Team Server folder.
- Tomcat is configured to use the ports 9080 and 9443 in file `JazzInstallDir/server/tomcat/conf/server.xml`. If necessary, change them as appropriate for your system. If necessary, also update the configuration settings of Jazz Team Server repository HTTP ports in the **Advanced Properties** configuration page in the Jazz administrative Web interface.
- The directory `JazzInstallDir/server/tomcat/logs` contains the server log files. If you have trouble starting the server, check the log files.

You are now ready to start the Tomcat server. To start the server, run the startup file.

1. To start the server as user root, run this command:

   ```
   ./server.startup
   ```

   If your user ID has administrator access, run this command:

   ```
   sudo ./server.startup
   ```

   Note: A separate Apache Tomcat console window is not visible. You can check the server startup progress by viewing the log file at `JazzInstallDir/server/tomcat/logs/catalina.out`.
2. To stop the server as user root, run this command:

   ```
   ./server.shutdown
   ```

   If your user ID has administrator access, run this command:

   ```
   sudo ./server.shutdown
   ```

   Note: This example runs the start and stop scripts directly from the directory `JazzInstallDir/server`.

What to do next

When the server is started, proceed to "Running the setup wizard."

Running the setup wizard:

Running the setup wizard verifies that the server is operating properly and guides you through the steps to configure the server.

About this task

To verify that the Jazz Team Server for System z is connecting to the database, look at the server log or console output. The connection and database information is echoed on its first access. The directory `JazzInstallDir/server/tomcat/logs` is used for the server log files.

This procedure assumes your server is available using the hostname *yourServerName* and the default port is *9443*. If necessary, replace *yourServerName* with your server hostname and replace port *9443*.

Start the setup wizard to configure your server. Use the URL: `https://yourServerName:9443/jazz/setup`.
The default user name and password are case-sensitive:

- If you use the local operating system (Linux for System z) as a security repository, log in as a *JazzAdmin* user that is defined in your Linux for System z system.
- The default user name is: ADMIN
- The default password is: ADMIN

Choose a setup path. The setup wizard has two main paths.

- The **Fast Path Setup** uses the default configuration. If you want to get the server running quickly, the fast path setup is a good option. During the Fast Path Setup, you set up the following:
  - user registry
- The **Custom Setup** guides you through the detailed server configuration, including the ability to enable e-mail notifications. During the Custom Setup, you set up the following:
  - database
  - e-mail notification
  - user registry

When the initial setup is complete, additional options can be configured from the Jazz Team Server Administrative Web user interface by using the URL: `https://yourServerName:9443/jazz/admin`.

What to do next

If the server setup wizard does not load, check the following items:

- Verify that the application server has started. Use the URL: `http://yourServerName:9080`.
- Verify the Jazz Team Server has started by logging in to the Jazz Team Server Administrative Web interface using the URL: `https://yourServerName:9443/jazz/admin`.
- The URI root for the Jazz Team Server path must be **`/jazz`**. For example: `https://example.com:9443/jazz` must be used rather than `https://example.com:9443`.

Completing the installation:

After the server is installed, you must consider some configuration options before continuing.

About this task

Some files contain passwords. Those files should be protected so that they are readable only by users authorized to know the password for the accounts.

-
  - **teamserver.properties** - The Jazz Team Server requires that the database password is stored in `JazzInstallDir/server/conf/jazz/teamserver.properties`.

    When properties files are saved, the application always makes a backup copy of the previous version in the same directory. If you want to remove all files that contain the clear-text password, remove the backup properties files after configuring the server for the first time.

- **cqconnector.properties** sync engine requires that user passwords be stored in an properties file.
- **cqconnector.properties** - The ClearQuest Connector requires that both ClearQuest® user IDs and passwords be stored in a properties file.

- When connecting to the server with the Rational Team Concert for System z client or a Web browser, you might see security certificate warnings. To disable the warning, see Disabling security certificate settings at http://publib.boulder.ibm.com/infocenter/rtc/v1r0m0/index.jsp?topic=/com.ibm.team.install.doc/topics/t_disable_server_certificates.html.
- You are now ready to connect to the server with the Rational Team Concert client or a Web browser.

Configuring Security-Enhanced Linux

If Security-Enhanced Linux (SELinux) is enabled, you must either disable it or change the security context of the Java Runtime Environments (JREs) used for installing and running Rational Team Concert to allow text relocation.

About this task

If you have set up a machine for the sole purpose of evaluating Rational Team Concert and the SELinux features are not important to you, then the easiest way to proceed is to disable SELinux.

Notes:

- SELinux is installed and enabled by default on Red Hat Enterprise Linux 5.
- SELinux is not installed on Suse Linux Enterprise Server 10.
- To disable SELinux from the command line:
  1. Run the **setup** command (this is/usr/bin/**setup**).
  2. Select **Firewall Configuration** and press **Enter**.
  3. Use the Tab and arrow keys to change the **SELinux** to Disabled.
  4. Select **OK** and press **Enter**.
- To change the security context of the JRE:
  1. In the `JazzInstallDir/server` directory, locate the directory `jre`.
  2. Run the command **chcon -R -t textrel_shlib_t** against the `jre` directory. This command recursively processes the files and allows text relocation. For example:

```
chcon -R -t textrel_shlib_t /opt/IBM/JazzTeamserver/server/jre
```

Setting up the Rational Build Agent

This section describes how to complete the installation of the Rational Build Agent on z/OS for Beta 2. You must have installed FMID HAHD200 as part of the SMP/E package installation of Rational Team Concert for System z prior to completing these steps. In addition, for Beta 2, you must mount the Rational Team Concert for System z installation HFS as writeable to complete these steps. After completing these steps, you can mount the installation HFS as read-only if you prefer.

Refer to Installing Rational Team Concert for System z for information about the FMIDs that can be installed independently from each other. The SMP/E package is contained in the download file rtcz-smpe-2.0beta2.zip, along with the Program Directory.

Installing and running the agent on z/OS

Follow these instructions to compile the Build Forge agent source code on z/OS. The agent source code for z/OS is provided as uncompiled source only. A binary distribution is not available.

The following software and programs are required:

- The c89 compiler and UNIX header files. On the z/OS system, the agent runs in the UNIX System Services (USS) environment.
- The z/OS UNIX shell interface. During installation, you run all commands on z/OS in the z/OS UNIX shell.

On the z/OS system, navigate to the bfagent/src directory under yourPathPrefix/usr/lpp/jazz (where yourPathPrefix is any path prefix specified during the SMP/E installation. Run the following command:

```
./configure-zos
```

After the ./configure-zos script completes, run the following command:

```
./build-zos
```

1. On the z/OS system, run the following commands to build the agent source code:

   ```
   cd yourPathPrefix/usr/lpp/jazz/bfagent/src
   ./configure-zos
   ```

   After the ./configure-zos script completes, run the following command:

   ```
   ./build-zos
   ```

   Note: If you receive errors after the ./build-zos script runs, see "Troubleshooting the Rational Build Agent installation on z/OS" on page 77.
2. Change to *yourPathPrefix*/usr/lpp/jazz/bfagent/src and run the following command from an authorized user:

   ```
   extattr +p -s bfagent
   ```
3. On the z/OS system, to start the agent manually, change to *yourPathPrefix*/usr/lpp/jazz/bfagent/src , and use the -s option:

   ```
   bfagent -s -f /etc/rtcz/bfagent.conf
   ```

   The agent runs as a standalone daemon and uses the default agent port 5555. To change the default port, use the port setting in bfagent.conf. See "bfagent Reference" on page 80. Also see inetd tips below.
4. On the z/OS system, use the telnet command to test the connection. See "Testing the connection" on page 78.

Tips for using inetd or xinetd

If the UNIX TCP/IP daemon (inetd or xinetd) is installed and active on the z/OS system, you can set up the Rational Build Agent to run as a service and start automatically. For additional information on configuring inetd, refer to the z/OS V1R9 Information Center at http://publib.boulder.ibm.com/infocenter/zos/v1r9/index.jsp?topic=/com.ibm.zos.r9.cs3/cs3.htm (or the appropriate information center for your version of z/OS). The full configuration of inetd is beyond the scope of this document. In a simple example, you could:

1. Modify /etc/inetd.conf by adding this line:

   ```
   bfagent stream tcp nowait userID /u/rtcz/bfagent -f /etc/rtcz/bfagent.conf
   ```

bfagent
Service name of the daemon. Default is bfagent (lowercase). The name must match the name used in /etc/services.

stream tcp nowait
Specific inetd configuration statements (socket type, protocol, wait flag). Do not modify.

User ID
User ID for the daemon process. The default is OMVSKERN. This user ID must be a user ID with a valid OMVS security segment, BPX.DAEMON permission and READ and EXECUTE permission to the installation and configuration directories.

/u/rtcz/bfagent
Server program (absolute location of bfagent). Default is `/u/rtcz/bfagent`. The arguments after this inetd argument are server arguments.

-f /etc/rtcz/bfagent.conf
Working directory (location of Build Forge server configuration file). The default is `/etc/rtcz/bfagent.conf`.

Important: Copy the customized Build Forge configuration files to a new directory (like /etc/rtcz/) to avoid overwriting them when applying maintenance. The working directory defined here must reflect this change.

2. Add the following to /etc/services:
   ```
   bfagent 5555/tcp #BUILD FORGE AGENT
   ```
3. Update the port in your bfagent.conf to map to your services entry:
   ```
   port 5555
   ```
4. Restart inetd

Running an agent

This section describes how to set up an agent to run. It is normally run as an auto-starting service or daemon, but for the Rational Team Concert for System z beta, you can start the bfagent program from the command line.

Configuring the agent

This section describes how to configure the agent after installation.

Locating the agent configuration file

The agent configuration file, bfagent.conf, provides runtime configuration of the agent's operation. It contains comments that explain all of the possible options. The file is located in the agent installation directory: *yourPathPrefix*/usr/lpp/jazz/bfagent/src. As specified previously, you can copy this to an alternate directory such as /etc/rtcz.

Important: If you make changes to the bfagent.conf file in the installation directory, you must repeat the changes after you subsequently reinstall or upgrade the agent. The configuration file is overwritten during every installation.

You can specify an alternate configuration file:

- You can preserve agent configurations by using a configuration file outside of the agent installation directory. When you do this, use the `-f` command line option on the command that starts the agent. Example:
  ```
  bfagent -s -f /etc/rtcz/bfagent.conf
  ```

Changing the agent port

If the agent is being installed on a server where port 5555 is already occupied, the agent port can be changed after it is installed.

To change the port:

1. Modify the port value in your bfagent.conf file.
2. If you are running a stand alone server (that is, you started with `bfagent -s`), you can you can use the `kill` command to stop the current bfagent process and restart using the command you used to start the agent the first time.
3. If you are running via inetd, you also need to modify your port setting in the /etc/services file and restart inetd.

Configuring a different shell

You can configure an agent to use a shell other than the default shell by editing parameters in the bfagent.conf file.

For example, to use the tcsh shell, you can set the shell parameter as follows:

```
shell /bin/tcsh
```

For more information on the bfagent.conf file see “bfagent.conf Reference” on page 80

Chapter 2. User scenarios for Rational Team Concert for System z

The user scenarios for the Rational Team Concert for System z beta program are designed to introduce you to basic Rational Team Concert functions and to provide early access to new System z components. In some cases, components that are still being developed are available as previews and might not be fully functional.

The following scenarios are included:

Scenario 1: Getting started
This scenario includes information for using the basic functions of Rational Team Concert and Rational Team Concert for System z.

Scenario 2: Using the System z Jazz Gateway to integrate with an existing SCM
Using SCLM (Software Configuration Library Manager) as an example, this scenario explains how to use the System z Jazz Gateway and your existing software configuration management system with Rational Team Concert for System z.

Scenario 3: Preview of the Mass Import Tool
This scenario provides early access to Rational Team Concert for System z mass import tool that is currently being developed.

Scenario 4: Using the Rational Build Agent to execute command driven builds
This scenario describes how to use Rational Team Concert for System z with the Rational Build Agent to run project builds.

Scenario 5: Using the Rational Build Agent and Job Monitor to execute builds using JCL
This scenario describes how to use Rational Build Agent and the Job Monitor to run project builds.

Scenario 6: Using the Rational Build Agent and Antz Build Extensions to compile a COBOL application
This scenario explains how to uses Ant to build a simple COBOL application on a z/OS build machine, use the Rational Team Concert for System z Data Definition, Translator, and Language Definition components to define the build environment, and create a shared zComponent project containing COBOL source code and a build.xml file that defines the Antz build process.

Scenario 1: Getting started with Rational Team Concert for System z

There are many resources to help you get started with Jazz components and Rational Team Concert for System z.

The Rational Team Concert for System z beta extends the functions of Rational Team Concert. Most of the documentation for Rational Team Concert V2 and V1 is appropriate for the System z version as well.

To help you get started, review the following resources:

Rational Team Concert help
The help content is available from the Rational Team Concert for System z client by selecting **Help** → **Help Contents** after you launch the client.

Online resources

The jazz.net community site and IBM developerWorks each have valuable information. You can find direct links from Rational Team Concert for System z client by selecting **Help → Web Resources** after you launch the client.

The information from these sources is relevant to this beta program; however, the following limitations apply:

- The Rational Team Concert for System z Beta 2 depends on Rational Team Concert V2 Milestone 3 driver, so any limitations apply to both.
- Rational Team Concert V2 is currently being developed; therefore some of the new features are not documented. Specifically, the tutorial does not reflect changes to the Iteration Plan editor or changes to the steps for creating repository workspaces.
- Rational Team Concert V2 information that applies to platforms that are not included in the Rational Team Concert for System z Beta 2 packages does not apply at this time.

The Rational Team Concert for System z team can assist with answering questions about how specific sections apply to the System z beta.

The tutorial, "Getting started with Rational Team Concert," is useful for introducing the basic concepts and terminology for Rational Team Concert and Rational Team Concert for System z. The tutorial uses Java projects and a Windows server, but the key concepts of Rational Team Concert for System z are still introduced.

In the section "Examine the Jazz Build," for the System z beta, you have the option to run a Jazz Build Engine on Windows or on z/OS. If you choose to run the Jazz Build Engine on Windows or Linux, you can install it on a Windows or Linux 32 bit machine using the launchpad for the Client for Eclipse IDE, Server and Optional Features package and run the jbe.exe (or jbe program on Linux) under the `jazz\buildsystem\buildengine\eclipse` directory where you installed the files.

If you prefer to run the Jazz Build Engine on z/OS instead of Windows as the tutorial instructs, see the additional information on customizing the Jazz Build Engine JCL in *Appendix A. Running the Jazz Build Engine on z/OS* later in this guide. The build section of the tutorial introduces basic concepts of Rational Team Concert for System z builds; however, the build agent for native z/OS builds is the Rational Build Forge agent as described in *Scenario 2: Using the Build Forge agent with Rational Team Concert for System z*.

Complete the basic tutorial before you work on the other user scenarios in this guide.

To start the tutorial, from the Rational Team Concert for System z client **Help** menu, select **Welcome → Tutorials → Get started with Rational Team Concert**. The tutorial will teach you how to:

- Set up a project and team.
- Get connected as a user.
- Create and understand work items.
- Organize team work.
- Save and share source.
- Use the Web user interface.

- Examine the Jazz build.

Expect to spend a few hours to complete the tutorial.

For a faster introduction, see the *Jazz technology platform quick reference* at https://jazz.net/jazzdocs/topic/com.ibm.team.platform.doc/topics/c_jazz-platform-quick-ref.html.

To get started using Rational Team Concert for System z, begin with the following help topics, which are available from the help menu of the Rational Team Concert for System z client or jazz.net:

- *Administering the Jazz Team Server through the Web interface*: https://jazz.net/jazzdocs/topic/com.ibm.team.repository.web.admin.doc/topics/tworkwithadminwebui.html
- *Working with projects, teams, and process*: https://jazz.net/jazzdocs/topic/com.ibm.team.platform.doc/topics/t_projects_teams_process.html

Explore the online help and the jazz.net Web site for product details, concepts, and troubleshooting, for Rational Team Concert, the Jazz platform, and new incubator technologies. After completing the getting started tutorial, the remaining scenarios in this guide will introduce you to the new functions of Rational Team Concert for System z.

Scenario 2: Using the System z Jazz Gateway to integrate with existing SCMs

With the System z Jazz Gateway, you can programmatically extract information from the Rational Team Concert for System z repository. Jazz already provides a REST interface to return work item information in the form of XML. The System z Jazz Gateway provides the interface that handles the secure user ID authentication and returns either the status of a work item or the complete XML.

This scenario uses SCLM, IBM's z/OS software configuration management (SCM) offering, to show how you can use the gateway as an interface to the Rational Team Concert for System z repository. SCLM has a number of user exit points where you can add your own processes. It is through the **Edit** and **Promote** user exit points in SCLM that you access the System z Jazz Gateway server. Additionally, SCLM can assign "Change Codes" to modules to tag the code as in process for a particular change. This change code maps to the Rational Team Concert for System z work item, so using the change code input field in SCLM to enter a Rational Team Concert for System z work item number we can perform the required checks.

Apply this scenario if you are using an existing z/OS SCM to manage your source code, such as SCLM, but want to use Rational Team Concert for System z to track defects and other tasks through work items. Using the Jazz Gateway, you can deny or allow processes to run on your z/OS by checking the current status of a work item. For example, if a specified work item does not exist, or is not **In Progress**, you can forbid edits. If the specified work item is not **Resolved**, you cannot move the code to the next stage of development.

Starting the System z Jazz Gateway server

About this task

To start the System z Jazz Gateway server, you must:

1. Configure the sample JCL member BLZGWSRV in *hlq*.SBLZSAMP, where *hlq* is the high level qualifier specified during the SMP/E installation.
2. Submit the modified JCL and check the job log. The following message must be in the STDOUT: `Gateway listening on port:3456` where 3456 is the port number you assigned in the BLZGWSRV member.

   Note: The gateway server must run in ASCII so ensure that you have specified `-Dfile.encoding=ISO8859-1`. This should be set by default in the job.

Setting the REXX Gateway client to communicate with the server

These steps describes how to set up and use the REXX Gateway client to communicate with the Jazz Gateway server.

About this task

Before you can use the System z Jazz Gateway server, you must configure the REXX Gateway client by completing the following steps:

1. Configure member BLZGWCLI, in *hlq*.SBLZSAMP, using the instructions in the sample REXX. The only parameters you need to set are the server address where the gateway is running, and the port number that it is monitoring.
2. To test the System z Jazz Gateway server, configure member BLZGWTST in hlq.SBLZSAMP data set, where you uploaded the samples, using the instructions in the sample REXX
3. To run the test enter `TSO EX hlq.SBLZSAMP(BLZGWTST)`. If you had set the test to return the work item status and if the work item exists, you will receive a message indicating the status of the work item similar to: `Workitem number 1 is in New status`. Note that this command is a sample and `hlq.SBLZSAMP` is the name of the data set where you uploaded the members. (Your data set name might be different.)

Results

If you set the test to return the full XML, you will receive the following message: `XML Returned for workitem number 1`. The view will change to the returned XML.

Setting up the System z Jazz Gateway to control SCLM workflow

SCLM has a number of user exit points that use the information returned by the System z Jazz Gateway server. This topic describes the steps to set up and use SCLM with the System z Jazz Gateway server.

Before you begin

You must have configured the REXX Gateway client as described in the previous step.

About this task

Complete the following steps to set up the System z Jazz Gateway:

1. If your SCLM project is not already invoking CCVFY and PROMOTE user exits, you can start them by modifying the FLMCNTRL macro and rebuilding the project definition. The FLMCNTRL macro will look something like this:

```
*
*********************************************************************
*               PROJECT CONTROLS
*********************************************************************
*
*
         FLMCNTRL ACCT=DOHERTL.ACCOUNT.FILE,                          C
               CCVFY=SCLMEXIT,               * CCVFY USER EXIT        C
               CCVFYCM=TSOLNK,               * METHOD TO CALL EXIT    C
               CCVFYDS=DOHERTL.PROJDEFS.SOURCE,                       C
               PRMVFY=SCLMEXIT,              * PROMOTE VERIFY USER EXITC
               PRMVFYCM=TSOLNK,              * METHOD TO CALL EXIT    C
               PRMVFYDS=DOHERTL.PROJDEFS.SOURCE,                      C
               MAXVIO=999999,                                         C
               VIOUNIT=VIO
*
```

2. The SCLMEXIT member below contains an invocation of the actual user exit. If your site already has SCLM user exits in place then you can add the `SELECT CMD ...` statement shown in the REXX code below into your existing user exits. If your site does not have SCLM user exits in place, then create a member in your project definition library that contains the code below, and make sure your user exit invocation statements point to it, as shown in the previous figure. You should change *BLZ.SBLZSAMP* to the data set name where you uploaded the samples.

```
/* REXX */
   ARG parm
   Address ISPEXEC
   "SELECT CMD(EX 'BLZ.SBLZSAMP(BLZSCLM1)' '"parm"') NEST"

Exit rc
```

3. Configure member BLZSCLM1 in the data set where you uploaded the samples, using the instructions in the sample REXX. This member will call the REXX Gateway client to check on the work item status. It will then allow or deny the SCLM action through the user exit return codes.

Using SCLM with Rational Team Concert for System z work items

The following test cases indicate the results that occur under the conditions set in the SCLM user exit code.

The terms ″change code″ and ″work item″ are interchangeable in these scenarios because the SCLM change code is passed through to Rational Team Concert for System z as a work item. The SCLM user exit checks the last change code saved against a member and uses that to check the status of a work item in Rational Team Concert for System z.

The SCLM user exit code uses an SCLM hierarchy of **DEV → TEST → PROD**.

Test case 1: Ensure that edit is not allowed if a change code is not entered

Instructions: Go to SCLM edit and select a member to edit, but do not enter a work item in the change code field.

Result: The user exit returns the following message: `You must enter a valid workitem number. Only numerics and no leading blanks or zeros.`

Test case 2: Ensure that edit is not allowed if a change code is entered that is not valid

Instructions: Go to SCLM edit, select a member to edit, but in the change code field, enter a work item that is not valid.

Result: The user exit returns the following message:

```
===> E: GWClient128E SOCKET(Read) rc=0 Error=Work Item 5555 does not exist.
Search for a different work item number. HTTP status is : 404.
The status of the socket set is GWClient Connected Free 39 Used 1
Workitem number 5555 does not exist in the RTC repository,
 or the gateway server is not running.
```

Test case 3: Ensure that edit is not allowed if a work item in a state other than "In Progress" is entered

Instructions: Go to SCLM edit, select a member to edit, then in the **Change Code** field, enter a valid work item that is in **Resolved** status.

Result: The user exit returns the following message: `Workitem number 96 is in Resolved status. This is not a valid status for edit.`

Test case 4: Ensure that edit is allowed if the work item number entered in the change code is in "In Progress" status

Instructions: Go to SCLM edit, select a member to edit, then in the **Change Code** field, enter a valid work item that is in **In Progress** status.

Result: Edit is allowed on the member.

Test case 5: Try to promote a member that has a change code assigned to it that is not in resolved status.

Instructions: Go to SCLM promote, select a member to promote that has been previously edited and has been assigned a change code that is a work item that is still in **In Progress** status.

Result: Promotion fails with the following message: `Workitem number 1 is in New status. This is not a valid status for promotion.`

Test case 6: Try to promote a member that has a change code assigned to it that is in resolved status.

Instructions: Go to SCLM promote, select a member to promote that has been previously edited and assigned a change code that is a work item that is in **Resolved** status.

Note: This involves going to Rational Team Concert for System z after the change in SCLM has been edited, built, and tested and is ready to be promoted. When the member is ready to be promoted, change the work item status to **Resolved**.

Result: Promotion is allowed to continue.

Test case 7: Try to promote a member from TEST to PROD when the work item is in resolved status.

Instructions: In this scenario the status of the work item must be **Closed** before promotion to PROD is allowed. Go to SCLM promote and select a member to promote that was previously promoted to TEST when the work item was in **Resolved** status. Try to promote that member to PROD while the work item is still in **Resolved** status.

Result: The promotion fails, and the following message is displayed:`You can only promote higher than TEST when status has moved from Resolved. This is not a valid status for promotion to PROD.`

Scenario 3: Preview of the mass import tool

The Rational Team Concert for System z beta gives you early access to some of the features and functions that are currently being developed. The mass import tool description is included in this scenario.

Overview of the Rational Team Concert for System z mass import tool

With the mass import tool, you can import members from multiple partitioned data sets (PDS) on a z/OS system into Rational Team Concert for System z source control. You can set up the import process using options and mapping files that automatically create the required project structure and data set definitions. You can also associate members with language definitions. A language definition describes how members are built by specifying preprocessing tasks and setting the correct compiler and compiler options.

The mass import tool is intended to be used by project administrators who are responsible for setting up version and source control systems.

The import process automatically creates a new repository workspace that contains projects, called *zComponent projects*, in which your PDS members are organized and stored. You can use a mapping file to define a more detailed project structure before you run the import command. With a mapping file, you can specify which PDS members should be placed in which zComponent projects and subfolders, called *zFolders*. The mapping file also specifies which component will contain each zComponent project. You can also use the mapping file to associate fully qualified PDS member names to a language definition that you previously created.

Run the mass import tool from the Rational Team Concert for System z command line interface (CLI) using the zimport subcommand of the SCM command. After you import the PDS members, they are contained by a zComponent project in the newly created repository workspace.

Beta limitations: The mass import tool is available as a preview in this beta program and its functions are not all available. Please note the following limitations:

- The zimport command runs on Windows or Linux operating systems. It does not run on z/OS at this time.
- You cannot import your PDS members to zComponents. Instead, you can create a PDS listing file to simulate the import process and create the zComponent project structure.

Importing z/OS partitioned data set members to a zComponent project

Before you begin

Before you can import z/OS partitioned data set (PDS) members to a Rational Team Concert zComponent project, the following conditions must be met:

- You must already have created a project area in Rational Team Concert for System z. If you prefer, after you create the zComponent project, you can also create a stream within that project area to be the default flow target for the repository workspace that the mass import command creates, though this is not

a required step. You can also designate language definitions to associate with your imported z/OS members, though this is not a required step either.

- The Rational Team Concert for System z server must be running when you begin a Rational Team Concert for System z mass import.
- For this Rational Team Concert for System z beta release, you must already have created a PDS list using your operating system text editor.

About this task

For this Rational Team Concert for System z beta release, the zimport command does not import an actual PDS from z/OS to the source control component. Instead, it operates from a simulated list of partitioned data sets that are contained in a file specified by the --pdslist argument. This function of using a simulated list is included in this beta to demonstrate the capability of the mapping file to map PDS names to the zComponent projects where they should be stored, and to the Jazz Components that will contain the zComponent projects. By the time of the final Rational Team Concert for System z v2.0 release, the zimport command will import a real PDS from z/OS. More information on command options and mapping file creation is found later in this section.

1. Enter the zimport command, which is a subcommand of scm. Your command line might look something like this:

   ```
   scm zimport --hlq SMITH --mapfile C:\mapping.txt
   --pdslist C:\listing.txt --projectarea TestProjectArea -s MyStream
   -r https://localhost:9443/jazz -u username -P password
   ```

   Tip: For more information on how to use the zimport command, run the scm help zimport command.

   Tip: If you want to see what the results of the zimport command would be without actually executing a mass import, specify either the -n or --dry-run option to scm, like this:

   ```
   scm -n zimport --hlq SMITH --mapfile C:\mapping.txt --pdslist C:\listing.txt
   --projectarea TestProjectArea -s MyStream
   -r https://localhost:9443/jazz -u username -P password
   ```

2. Specify the -r option to indicate which Rational Team Concert for System z repository to import the PDS members to.
3. Specify the -u option to indicate a user name for the repository.
4. Specify the -P option to indicate a password for the repository.
5. Specify the --hlq option to indicate the high level qualifier of the partitioned data sets whose members you want to import.
6. Specify the --mapfile option to indicate the location of the mapping file the mass import tool should use.
7. Specify the --projectarea option to indicate the name of the project area where you want to store the data set definitions, and where the language definitions are stored.
8. **Optional:** Specify the -s option to indicate the name, alias, or universal unique identifier (UUID) of the stream to set as the default flow target for the repository workspace that the zimport command creates.
9. **Optional:** Specify the -q option to suppress all console output.
10. **Optional:** Specify the -v option to increase the verbosity of the output.

Note: If you specify both -q and -v, no output is displayed.

What to do next

After the zimport process is complete, a new Rational Team Concert for System z repository workspace exists, which contains components that contain zComponent projects, which themselves contain zFolders and zFiles that correspond to the imported PDS and its members. If you specify the -s option, your new workspace will use that default flow target. You should also see new data set definitions in the Rational Team Concert for System z Team Artifacts view.

Tip: You might have to refresh the list before these new data set definitions display.

Rational Team Concert for System z will create a data set definition for each PDS you import, and the corresponding zFolder will be associated with that data set definition, too. If you specified any language definition mappings, the zFiles should also be associated with those specified language definitions.

Options supported by the Rational Team Concert for System z mass import tool

This topic lists and briefly describes the options available through the mass import tool.

Run the mass import command, zimport, from the Rational Team Concert for System z command line interface (CLI). The mass import tool supports the following options:

-r For *repositoryURI*. Use this option to specify the location of the repository to which you want to import the partitioned data set (PDS) members.

-u For *user*. Use this option to specify a user name for the repository.

-P For *password*. Use this option to specify a repository password.

Important: Use a capital P for this option, not a lower case p.

--hlq For *HLQ*. Use this option to specify the high level qualifier (HLQ) of the partitioned data sets you want to import.

--mapfile

For *file*. Use this option to specify the location of the mapping file to use. This mapping file describes the structure of the project to which you want to import the PDS members. You must always specify the location of the mapping file, and your mapping file must contain the following two types of mapping:

- Mapping from the PDS members to the zComponent project to which you want to import them.
- Mapping from the zComponent project to the Jazz component to which the project belongs.

It is **optional** for your mapping file to contain mapping from members to their associated language definition.

Remember: The language definition must already exist in the specified project area.

--projectarea
For *projectArea*. Use this option to specify the name of the project area where the data set definitions that are related to the imported members should be stored. This should be the same project area where the related language definitions are stored.

-s **Optional:** For *stream*. Use this option to specify the name, alias, or UUID of the stream you want to set as the default flow target for the repository workspace that is created with your mass import.

-q **Optional:** For *quiet*. Use this option to suppress all console output.

-v **Optional:** For *verbosity*. Use this option to increase the output verbosity.

Mapping file name format

The mapping file name is line-delimited, wherein each line must follow this format: <identifier character> : <key> = <value> The table below indicates possible line combinations. For more information about mapping file rule restrictions, see "Rational Team Concert for System z mass import tool: mapping file rule restrictions" on page 42.

Table 1.

Identifier	Key	Value	Notes
C	zComponent project name	Jazz component name	Specifies to which Jazz Component a zComponent project should belong. zComponent projects that do not match project name rules specified in the mapping file cannot be imported.
P	Fully qualified member name (not including HLQ)	zComponent project name (data set name)	Specifies to which zComponent project a PDS should be imported. You can use ″wildcards″ (*) in the key to specify multiple members to put into the same zComponent project. Any member that does not match a member name rule specified in the mapping file cannot be imported.

Table 1. (continued)

Identifier	Key	Value	Notes
L	Fully qualified member name (not including HLQ)	Language definition name	Specifies a language definition to associate with imported members. Remember: The language definition must already exist in the project area. You can use wildcards in the key, indicated by an asterisk (*), to specify multiple members to associate with a language definition.

Important: A note about the *P* identifier: By default, zimport imports a PDS to the specified zComponent project as a data set with the same name as the PDS. Optionally, you can change the name of the data set by adding a colon (:) after the name of the zComponent project, followed by a data set name of your choosing. For example, if you want to import a PDS named *MORT.BLD.TEST* to zComponent project *MortgageApp* as data set *MORT.BLD*, the mapping file name should be *P:MORT.BLD.TEST=MortgageApp:MORT.BLD*.

You can also make *back* references in any line in the mapping file name by using parentheses in the key and *%n* in the value. For example, if you want to import all the members in a PDS named *MORT.*.BLD.TEST* to a zComponent project named *MORTApp* in a data set named whatever is between *MORT.* and *.BLD*, you could write the rule as *P:(MORT).(*).BLD.TEST.*=%1App:%2*.

Required formatting of a partitioned data set list file

For this Rational Team Concert for System z beta release, you must create a partitioned data set (PDS) list file to which you must point the Rational Team Concert for System z mass import tool so that you can import a PDS on that list to a zComponent project.

The PDS list file is a line-delimited file of PDS names, including high level qualifiers, that simulates a list of partitioned data sets (and the members they contain) that are available to be imported from Multiple Virtual Storage (MVS). The format of each line must be as follows:

<pds name>:<member1>,<member2>, and so forth

The contents of the PDS list file should follow a format that is similar to these examples:

- SMITH.HELLO.COBOL:HELLO
- SMITH.HELLO.LINK:HELLO
- SMITH.HELLO.OBJ:HELLO
- JONES.MORTGAGE.BLD.BMS:EPSMLIS,EPSMORT
- JONES.MORTGAGE.BLD.COBOL:EPSCMORT,EPSCSMRD,EPSCSMRT,EPSMDRVR

- JONES.MORTGAGE.BLD.JCL:DEFMORT,EPSCICS,EPSCMORT,EPSCSMRD,EPSCSMRT
- JONES.MORTGAGE.BLD.LINK:EPSCMORT,EPSCSMRD,EPSCSMRT,EPSMLIS
- JONES.MORTGAGE.BLD.OBJ:EPSCMORT,EPSCSMRD,EPSCSMRT,EPSMLIS,EPSMLIST
- JONES.MORTGAGE.BLD.SYSDEBUG:EPSCMORT,EPSCSMRT,EPSMLIST,EPSMPMT
- JONES.ALL.COBOL:EP1CALC,EP2CALC,EP3CALC,EP4CALC,EP5CALC,ATMC1,ATMC2

An overview of creating a mapping file to use with the Rational Team Concert for System z mass import tool

Before you can use the zimport command to perform a Rational Team Concert for System z mass import, you must create a mapping file to define a list of rules that tell the zimport command three things it needs to know to run successfully:

- which zComponent project and zFolder to put members of a partitioned data set (PDS) into
- which Jazz component should contain that zComponent project
- which language definition to associate with the PDS members

  Note: You do not have to associate a language definition with imported members. This is an optional step.

For a list and descriptions of these rules, see "Rational Team Concert for System z mass import tool: mapping file rule restrictions."

Rational Team Concert for System z mass import tool: mapping file rule restrictions

The rules in a mapping file are line-delimited, and the following restrictions apply to mapping file rules:

- The mass import tool ignores blank lines.
- The mass import tool ignores lines that begin with the pound sign (#).
- Lines must adhere to the following format, where *rule-type* must either be the character *P*, the character *C*, or the character *L*:

  rule-type:key=value

  – *P* indicates that the rule maps a qualified member name to the zComponent project and, optionally, the zFolder where it should be placed.

    Note: After the *P* rule, you have the option of specifying a zFolder to which to import the member. To do this, add *[:zFolder]* after the *P* rule, but replace *zFolder* with the name of the folder where you want to import the member. Such a line might look similar to this:
    `P:TEST.COBOL.HELLO=MyProject:MyFolder`
    Attention: The qualification of the member name in the *P* rule does not include the high level qualifier (HLQ).

  – *C* indicates that the rule maps a zComponent project to the Jazz component that should contain it.

    Important: If the zComponent project into which members are imported does not match any *C* rules, those members will not be imported.

– *L* indicates that the rule maps a qualified member name (without a high level qualifier) to the name of a language definition to associate with it.

  Important: The specified language definition must already exist in the project area that you specified on the command line. Specifying language definitions for imported members is optional, and members without an associated language definition can still be imported.

The contents of a mapping file might look like the following example:

```
# Member to zComponent project mappings
# Specify the zComponent project and, optionally, the zFolder which will contain the imported memb
#
# Format:
# P:<member>=<zComponent project>[:<zFolder>]
P:MORTGAGE.*.(*).*=MortgageApp:%1
P:ALL.COBOL.*CALC=CalcApp:CobolSrc
P:ALL.COBOL.ATM*=ATMApp:CobolSrc
# zComponent project to Jazz Component mappings
# Specify the Jazz Component that each zComponent project will be shared to.
#
# Format:
# C:<zComponent project>=<Jazz Component>
C:MortgageApp=Mortgage
C:CalcApp=Sample Applications
C:ATMApp=Sample Applications

# Member to Language Definition mappings
# Optionally specify the Language Definition to associate with imported members.
#
# Format:
# L:<member>=<Language Definition name>
L:*.COBOL.*=COBOL
L:*.JCL.*=JCL
```

If you run the zimport command for a PDS named `SMITH.MORTGAGE.BLD.COBOL`, indicate `SMITH` as the --hlq, and specify the mapping file rule `P:MORTGAGE.BLD.COBOL.*=MortgageApp`, all of the contents of `SMITH.MORTGAGE.BLD.COBOL` would be imported into a zComponent project named *MortgageApp*, inside a zFolder named *MORTGAGE.BLD.COBOL*. If, instead, you specify mapping file rule `P:MORTGAGE.BLD.COBOL.*=MortgageApp:CobolSRC`, the zFolder would be named *CobolSRC*.

Scenario 4: Using the Rational Build Agent to execute command driven builds

The purpose of this scenario is to introduce how Rational Team Concert for System z uses the Rational Build Agent for building native z/OS artifacts using existing commands. This scenario does not depend on the Job Monitor or other libraries. This scenario requires the successful installation of FMID HAHD200 (Rational Build Agent) as part of the SMP/E installation. If you want to use the example REXX exec from this scenario, it is installed as part of FMID HAHB200 (Build System Toolkit for System z)

In this beta, the agent is unchanged from the existing Rational Build Agent. The new function provided by Rational Team Concert for System z enables you to define and run builds that use the Rational Build Agent. This functionality is being developed and will be extended in future beta programs. The Jazz Build Engine is still supported by Rational Team Concert for System z.

Defining and running a build using the Rational Build Agent

Before you begin

You must already have created a project area in Rational Team Concert for System z.

About this task

This task is relevant if you are adding a build definition to your project area and you want to use the Rational Build Agent. The scenario assumes you have an existing project area and you want to add a build definition that uses the Rational Build Agent. This scenario uses a REXX exec to compile sample COBOL members. You can use other commands if you prefer.

To complete the user scenario, follow these steps:

1. If you have not already started the Rational Build Aent, start it now using the `bfagent -s` command as mentioned previously in the section on setting up the Rational Build Agent.

   Important: Commands executed through the Rational Build Agent run with the authority of the user that starts the agent.
2. Locate the BLZCSAMP member in hlq.SBLZSAMP, where hlq is the high level qualifier used during the SMP/E installation of Rational Team Concert for System z. Note that this member is actually installed as part of the installation of FMID HAHB200.. The BLZCSAMP member provides a sample REXX exec to complete a COBOL compilation. Review the comments in this sample member to understand the parameters and possible customization needed for your environment.

Setting up and running a build definition

1. In the Team Artifacts view, right click on the **Build Engines** icon and select **New Build Engine**.
2. Specify the Build engine ID as **RationalBuildAgent**, choose a Project or Team Area, and click **Save**.
3. In the Team Artifacts view, expand the project area folder in which you want to create a build definition.
4. Right-click **Builds** → **New Build Definition**.
5. In the New Build Definition window, deselect **Pre Build Command line** and **Post Build Command line**, select **Create a new build**, then click **Next**.
6. In the General Information window, enter a build definition ID and a brief description of the build definition.

   Important: Select **Rational Build Agent** from the Available Templates menu. Click **Next**.
7. In the Additional Configuration window, select both **General**, **Properties**, and **Rational Build Agent**, and then click **Finish**. The build definition you created opens in the Build Definition editor.

Performing a connection test with the Rational Build Agent

1. Click the Build Agent tab you created in the previous steps.
2. Enter the following information to connect to Build Forge:

   Hostname
   The name of the system that uses the Rational Build Agent.

Port
The port that communicates with Rational Build Agent. The default is port 5555.

User name
The name of the user who is connecting to the Rational Build Agent.

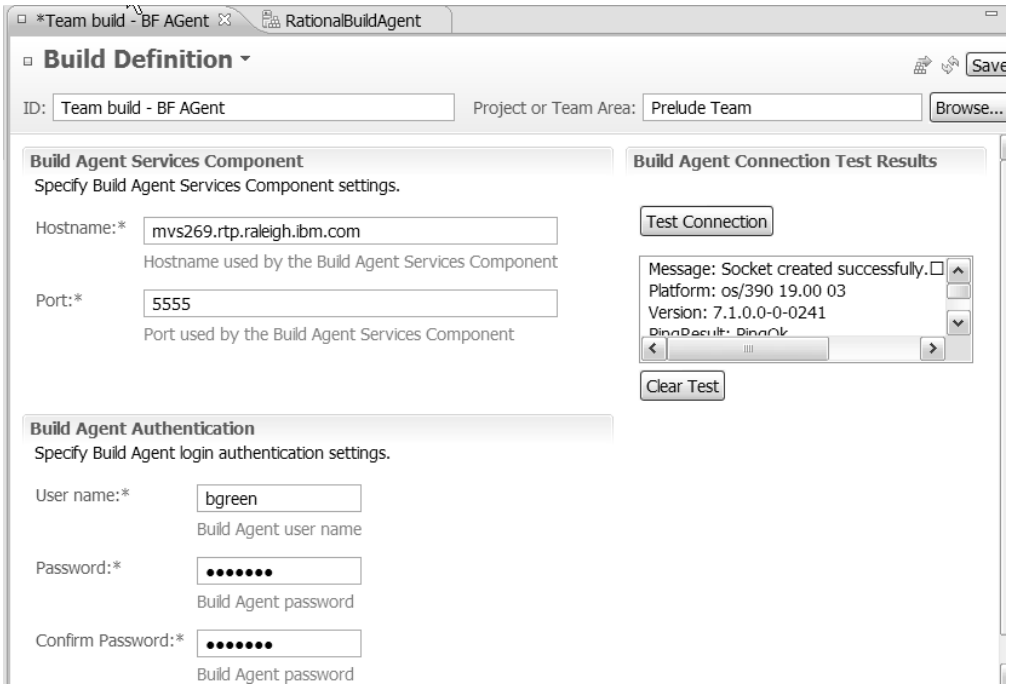
Note: This user name is used to authenticate the connection to the system, it does not have to be the same user name that started the agent.

Password
The password for the given user name.

Confirm Password
Enter again the password for the given user name.

3. Click **Test Connection**. The results of the connection test are displayed in the Rational Build Agent Connection Test Results box.

Example



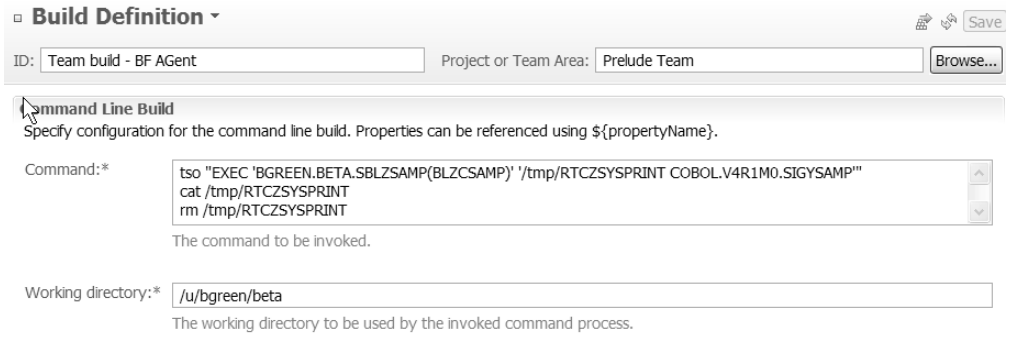
Defining a command block for the Rational Build Agent to run

1. Click the Build Agent Command Line tab.
2. In the Command field, enter a process command to send to the Rational Build Agent.
3. In the Working directory field, enter the working directory you want the invoked process to use.
4. Click **Save** to save the build definition.

Tip: The first time you save a build definition, a new build engine called "RationalBuildAgent" will be defined for the project area. Make sure that the "RationalBuildAgent" is the selected build engine on the **General** tab of the build definition.

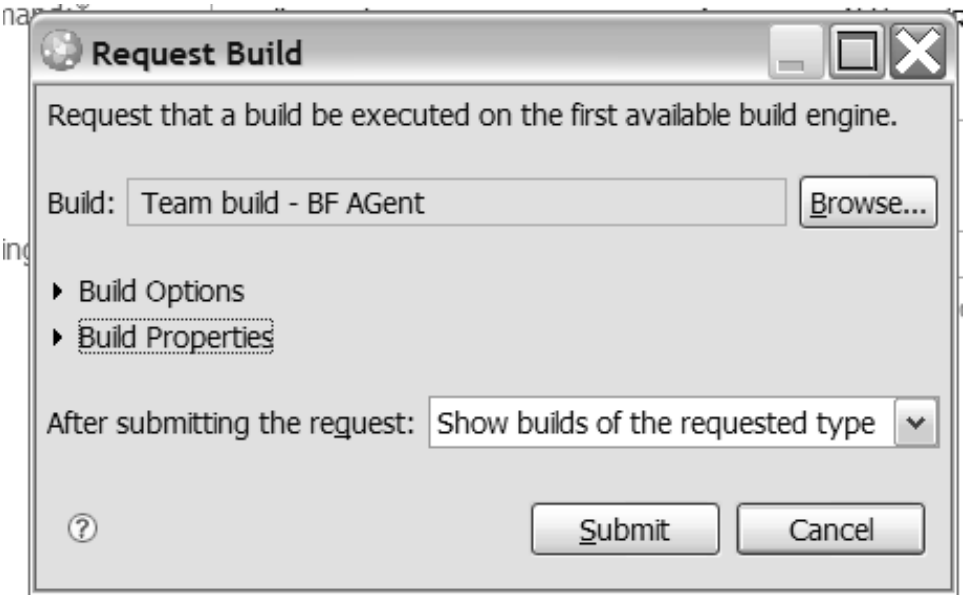
Example

To run the provided sample, you can set up a command block similar to the following example:



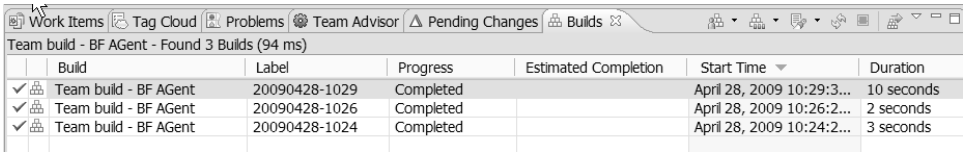
Requesting a build

1. In the Team Artifacts view, right-click the build definition, then select **Request build**. Alternately, you can click the Request Build icon in the upper right corner of the Build Definition window. The Request Build window opens.
2. Click **Submit**.



The Builds window opens.

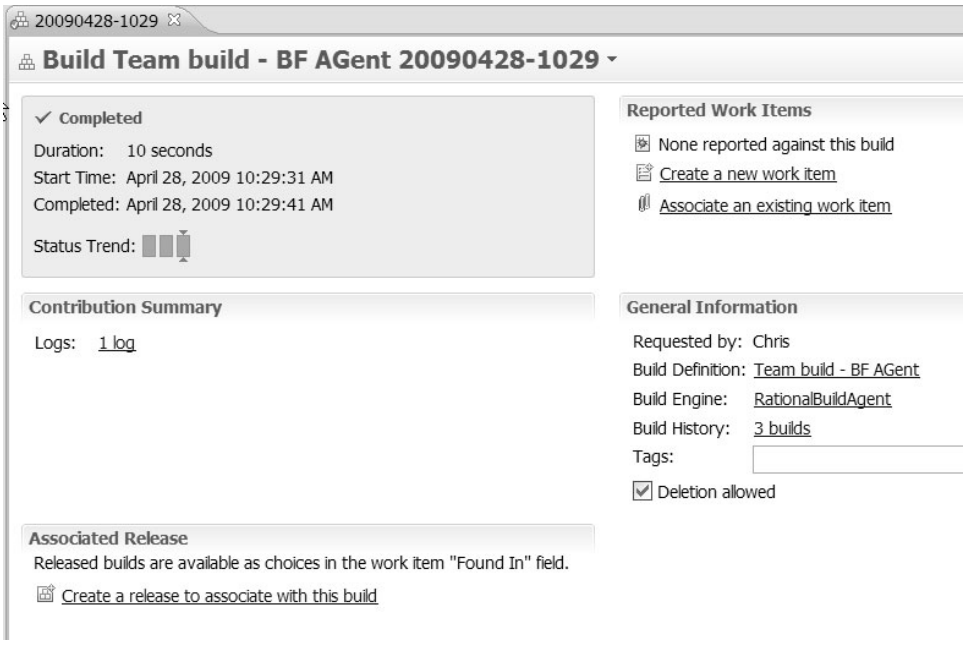
Tip: To monitor the progress of your build while it is processing, click the refresh icon (circular arrows).



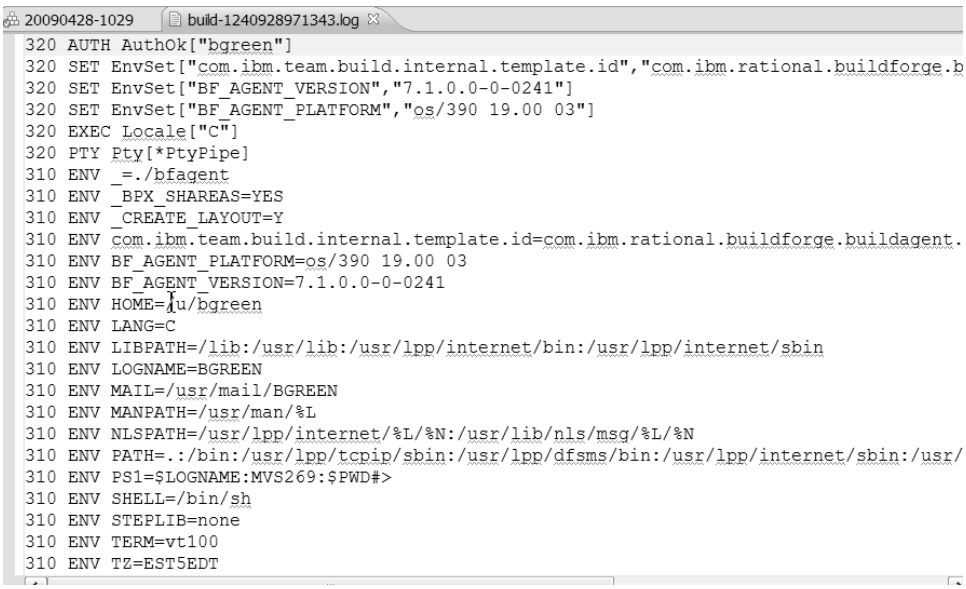
Example

Checking the build results

1. After the build is complete, double-click your build in the **Builds** tab to view the build results.



2. In the Contribution Summary section, click the **log** to view the Logs tab.
3. Select the log and click **Open** to display the detailed build results, including Rational Build Agent information and the results of the command block you entered.



Scenario 5: Using the Rational Build Agent and Job Monitor to execute builds using JCL

The purpose of this user scenario is to show how to use the Rational Team Concert for System z components to submit a build for native z/OS artifacts and obtain results using the Job Monitor. The first section of this scenario describes the customization of the Job Monitor component, followed by an example of using Rational Team Concert for System z and the Rational Build Agent to submit JCL for a z/OS build. This scenario depends on the successful SMP/E installation of FMID HAHC200 (Job Monitor) and HAHD200 (Rational Build Agent).

Note: If you have an existing instance of the Rational Developer for System z Job Monitor on your system, it can be used for Job Monitoring in this scenario.

Job Monitor Customization

Before you begin

The Job Monitor component depends on the SMP/E installation of FMID HAHC200. You will need the assistance of a security administrator and a TCP/IP administrator to complete this customization task, which requires the following resources and special customization tasks:

- APF authorized data set
- Various security software updates
- TCP/IP port for internal communication

About this task

In order to verify the installation and to start using Job Monitor at your site, you must perform the following tasks. Unless otherwise indicated, all tasks are mandatory.

1. Define an APF authorized data set. For details see "PARMLIB changes."
2. Create a started task procedure. For details see "PROCLIB changes" on page 49.
3. Customize the Job Monitor configuration files. For details see "BLZJCNFG, Job Monitor configuration file" on page 49.
4. Update security definitions. For details see Appendix F, "Job Monitor security," on page 95.

PARMLIB changes

Refer to *MVS™ Initialization and Tuning Reference (SA22-7592)* for more information on the PARMLIB definitions listed below. Refer to *MVS System Commands (SA22-7627)* for more information on the sample console commands.

APF authorizations in PROGxx

In order for Job Monitor to access JES spool files, module `BLZJMON` in the *`hlq`*`.SBLZAUTH` load library, where *hlq* is the high level qualifier used during SMP/E installation, and the Language Environment® (LE) runtime libraries `(CEE.SCEERUN*)`must be APF authorized.

APF authorizations are defined in `SYS1.PARMLIB(PROGxx)`, if your site followed IBM® recommendations.

APF authorizations can be set dynamically (until the next IPL) with the following console commands, where volser is the volume on which the data set resides if it is not SMS managed:

- SETPROG APF,ADD,DSN=*hlq*.SBLZAUTH,SMS
- SETPROG APF,ADD,DSN=CEE.SCEERUN,VOL=volser
- SETPROG APF,ADD,DSN=CEE.SCEERUN2,VOL=volser

PROCLIB changes

The started task listed below must reside in a system procedure library defined to your JES subsystem. In the instructions below, the IBM default procedure library, SYS1.PROCLIB, is used.

Customize the sample started task member *hlq*.SBLZSAMP(BLZJJCL), as described within the member, and copy it to SYS1.PROCLIB. As shown in the code sample below, you have to provide the following:

- The high level qualifier of the (authorized) load library, default BLZ. Replace BLZ with your *hlq*.
- The Job Monitor configuration file, default *hlq*.SBLZSAMP(BLZJCNFG).

```
//*
//* JOB MONITOR
//*
//JMON     PROC PRM=,             * PRM='-TV' TO START TRACING
//            LEPRM='RPTOPTS(ON)',
//            HLQ=BLZ,
//            CFG=BLZ.SBLZSAMP(BLZJCNFG)
//*
//JMON     EXEC PGM=BLZJMON,REGION=0M,TIME=NOLIMIT,
//            PARM=('&LEPRM,ENVAR("_CEE_ENVFILE=DD:ENVIRON")/&PRM')
//STEPLIB  DD DISP=SHR,DSN=&HLQ..SBLZAUTH
//ENVIRON  DD DISP=SHR,DSN=&CFG
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//         PEND
//*
```

BLZJCNFG, Job Monitor configuration file

Job Monitor provides JES related services. The behavior of Job Monitor can be controlled with the definitions in BLZJCNFG.

Customize the sample Job Monitor configuration member *hlq*.SBLZSAMP(BLZJCNFG), as shown in the following sample. Comment lines start with a pound sign (#), when using a US code page. Data lines can only have a directive and its assigned value, comments are not allowed on the same line.

Note: The BLZJMON started task must be restarted to pick up any changes you make.

```
HOST_CODEPAGE=IBM-1047
SERV_PORT=6716
TZ=EST5EDT
#_BPXK_SETIBMOPT_TRANSPORT=TCPIP
#CODEPAGE=UTF-8
#CONCHAR=$
#CONSOLE_NAME=JMON
#GEN_CONSOLE_NAME=OFF
#LIMIT_COMMANDS=NOLIMIT
#LIMIT_VIEW=USERID
#LISTEN_QUEUE_LENGTH=5
#MAX_DATASETS=32
```

```
#MAX_THREADS=200
#TIMEOUT=3600
#TIMEOUT_INTERVAL=1200
#SUBMITMETHOD=TSO
#TSO_TEMPLATE=BLZ.SBLZSAMP(BLZTSO)
```

HOST_CODEPAGE
The host codepage. The default is `IBM-1047`. Change to match your host codepage.

SERV_PORT
The port number for Job Monitor host server. The default port is `6716`. Change as desired.

Note: Before selecting a port, verify that the port is available on your system with the TSO commands **NETSTAT** and **NETSTAT PORTL**.

TZ Time zone selector. The default is `EST5EDT`. The default time zone is UTC +5 hours (Eastern Standard Time (EST) Eastern Daylight Savings Time (EDT)). Change this to represent your time zone. Additional information can be found in the *UNIX System Services Command Reference (SA22-7802)*.

The following definitions are optional. If omitted, default values will be used as specified below:

_BPXK_SETIBMOPT_TRANSPORT=<tcpip stack name>
Specifies the name of the TCPIP stack to be used. The default is `TCPIP`. Uncomment and change to the requested TCPIP stack name, as defined in the `TCPIPJOBNAME` statement in the related TCPIP.DATA.

Note: Coding a SYSTCPD DD statement in the server JCL does not set the requested stack affinity.

CODEPAGE
The workstation codepage. The default is `UTF-8`. The workstation codepage is set to `UTF-8` and generally should not be changed. You might need to uncomment the directive and change `UTF-8` to match the workstation's codepage if you have difficulty with NLS characters, such as the currency symbol.

CONCHAR
Specifies the JES console command character. `CONCHAR` defaults to `CONCHAR=$` for JES2, or `CONCHAR=*` for JES3. Uncomment and change to the requested command character

CONSOLE_NAME
Specifies the name of the EMCS console used for issuing commands against jobs (Hold, Release, Cancel and Purge). The default is `JMON`. Uncomment and change to the desired console name, using the guidelines below.

- `CONSOLE_NAME` must be either a console name consisting of 2 to 8 alphanumeric characters, or '`&SYSUID`' (without quotes).
- If a console name is specified, a single console by that name is used for all users. If the console by that name happens to be in use, then the command issued by the client will fail.
- If `&SYSUID` is specified, the client user ID is used as the console name. Thus a different console is used for each user. If the console by that

name happens to be in use (for example, the user is using the SDSF ULOG), then the command issued by the client might fail, depending on the GEN_CONSOLE_NAME setting.

No matter which console name is used, the user ID of the client requesting the command is used as the LU of the console, leaving a trace in syslog messages IEA630I and IEA631.

```
IEA630I OPERATOR console NOW ACTIVE,   SYSTEM=sysid, LU=id
IEA631I OPERATOR console NOW INACTIVE, SYSTEM=sysid, LU=id
```

GEN_CONSOLE_NAME

Enables or disables automatic generating of alternative console names. The default is OFF. Uncomment and change to ON to enable alternative console names.

This directive is only used when CONSOLE_NAME=&USERID and the user ID is not available as console name.

If GEN_CONSOLE_NAME=ON, an alternative console name is generated by appending a single numeric digit to the user ID. The digits 0 through 9 are attempted. If no available console is found, the command issued by the client fails.

If GEN_CONSOLE_NAME=OFF, the command issued by the client fails.

Note: The only valid settings are ON and OFF.

LIMIT_COMMANDS

Defines against which jobs the user can issue selected JES commands (Show JCL, Hold, Release, Cancel, and Purge). The default (LIMIT_COMMANDS=USERID) limits the commands to jobs owned by the user. Uncomment this directive and specify LIMITED or NOLIMIT to allow the user to issue commands against all spool files, if permitted by your security product.

Table 2. LIMIT_COMMANDS command permission matrix

	Job owner	
LIMIT_COMMANDS	**User**	**Other**
USERID (default)	Allowed	Not allowed
LIMITED	Allowed	Allowed only if explicitly permitted by security profiles
NOLIMIT	Allowed	Allowed if permitted by security profiles or when the JESSPOOL class is not active

Note: Note: The only valid settings are USERID, LIMITED, and NOLIMIT.

LIMIT_VIEW

Defines what output the user can view. The default (LIMIT_VIEW=NOLIMIT) allows the user to view all JES output, if permitted by your security product. Uncomment this directive and specify USERID to limit the view to output owned by the user.

Note: The only valid settings are USERID and NOLIMIT.

LISTEN_QUEUE_LENGTH

The TCP/IP listen queue length. The default is 5. Do not change unless directed to do so by the IBM support center.

LISTEN_QUEUE_LENGTH
The TCP/IP listen queue length. The default is 5. Do not change unless directed to do so by the IBM support center.

MAX_DATASETS
The maximum number of spooled output data sets that Job Monitor will return to the client (for example, SYSOUT, SYSPRINT, SYS00001, and so on). The default is 32. The maximum value is 2147483647.

MAX_THREADS
Maximum number of users that can be using one Job Monitor at a time. The default is 200. The maximum value is 2147483647. Increasing this number may require you to increase the size of the Job Monitor address space.

TIMEOUT
The length of time, in seconds, before a thread is killed due to lack of interaction with the client. The default is 3600 (1 hour). The maximum value is 2147483647. TIMEOUT=0 disables the function.

TIMEOUT_INTERVAL
The number of seconds between timeout checks. The default is 1200. The maximum value is 2147483647.

SUBMITMETHOD=TSO
Submit jobs through TSO. The default (SUBMITMETHOD=JES) submits jobs directly into JES. Uncomment this directive and specify TSO to submit the job through TSO **SUBMIT** command. This method allows TSO exits to be invoked; however, it has a performance drawback and for that reason it is not recommended.

Note:

1. The only valid settings are TSO and JES.
2. If SUBMITMETHOD=TSO is specified, then TSO_TEMPLATE must also be defined.

TSO_TEMPLATE
Wrapper JCL for submitting the job through TSO. The default value is *hlq*.SBLZSAMP(BLZTSO). This statement refers to the fully qualified member name of the JCL to be used as a wrapper for the TSO submit. See the SUBMITMETHOD statement for more information.

Note:

1. A sample wrapper job is provided in BLZ.SBLZSAMP(BLZTSO). Refer to this member for more information on the customization needed.
2. TSO_TEMPLATE has no effect unless SUBMITMETHOD=TSO is also specified.

Submitting JCL inside the build command

The Rational Build Agent allows you to submit JCL to JES. An agent running on z/OS can monitor and report build results by communicating with an instance of the Job Monitor running on the same z/OS system.

Rational Build Agent supports JCL submission through the .submitJCL command. This scenario will demonstrate how to submit JCL contained in a build system data set, as well as how to provide the JCL within a Build Definition.

Prerequisites

The Rational Build Agent must be running on the target build system before beginning this scenario. The Job Monitor must also be running on the same build system as the agent.

In order to communicate with the Job Monitor, the Build Agent must have the **jcl_submit_user** and **job_monitor_port** parameters set to appropriate values in the bfagent.conf configuration file.

The **jcl_submit_user** parameter is used to provide the credentials that will be used by Job Monitor when submitting jobs to JES. Add the following line to the bfagent.conf file:

```
jcl_submit_user userid:encrypted_password
```

where *userid* is the system id of the user for submitting jobs, and *encrypted_password* is an encrypted version of that user's password. The encrypted form of the password can be found by executing the following line at the USS command prompt:

```
bfagent –e password
```

where *password* is the password to be encrypted. The command will print a text string containing the encrypted value. The result will be similar to the following:

```
050405aaeb43166a00f763716b989f26651e2448ce309b72680a
```

The **job_monitor_port** parameter is used to specify the port to which Job Monitor is listening. Add the following line to the bfagent.conf file:

```
job_monitor_port XXXX
```

where *XXXX* is the Job Monitor port. This port should match your **SERV_PORT** setting for Job Monitor, which is set to 6716 in the *hlq*.SBLZSAMP(BLZJNCFG) file.

If an instance of the Rational Build Agent is running, you must restart the agent prior to continuing with this scenario.

Submitting JCL Contained in a Build System Data Set

JCL contained in a data set on the target build system can be submitting using the Rational Build Agent. The Job Monitor will submit the job to JES and report the results of the request. Build results can then be viewed through the Rational Team Concert for System z client.

1. Create a data set member containing the following JCL. Note that this job contains inline COBOL source code that will be compiled and link-edited. Customize the data set names contained in this job to values appropriate to your target system.

```
//HELLO    JOB ,NOTIFY=DEARTH
//*
//* COBOL COMPILATION
//*
//COBOL    EXEC PGM=IGYCRCTL,PARM='NODECK,OBJECT,LIB'
//STEPLIB  DD DSN=COBOL.V4R1M0.SIGYCOMP,DISP=SHR
//SYSIN    DD *
       IDENTIFICATION DIVISION.
       PROGRAM-ID. HELLO.
       PROCEDURE DIVISION.
       MAIN.
           DISPLAY 'HELLO, RTCZ.'.
```

```
          STOP RUN.
/*
//SYSLIN   DD DSN=DEARTH.SAMPLE.OBJ(HELLO),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT3   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT4   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT5   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT6   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT7   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//*
//LINKEDIT EXEC PGM=IEWBLINK,PARM='LIST,LET,MAP,XREF,REUS,RENT'
//SYSLIN   DD *
 INCLUDE SYSLIB(HELLO)
 NAME HELLO(R)
/*
//SYSLIB   DD DSN=DEARTH.SAMPLE.OBJ,DISP=SHR
//         DD DSN=CEE.SCEELKED,DISP=SHR
//SYSLMOD  DD DSN=DEARTH.SAMPLE.LOAD(HELLO),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//*
```

2. Create a Build Definition using Rational Team Concert for System z client.
   a. In the Team Artifacts view, right click on the Build Engines icon and select **New Build Engine**.
   b. Specify the **Build Engine ID** as RationalBuildAgent, choose a Project or Team Area, and click **Save**.
   c. In the Team Artifacts view, right click on the Builds icon and select **New Build Definition**.
   d. Click **Next**.
   e. Specify a Build Definition ID and select Rational Build Agent as the build template.
   f. Click **Next**.
   g. Clear the **Pre Build Command line** checkbox and click Next.
   h. Clear the Post Build Command line checkbox and click **Finish**.
   i. On the **Overview** tab, select RationalBuildAgent as the **Supporting Build Engine**.
   j. The build agent tab should contain the following values:
      1) Hostname: Your build machine's IP address or hostname.
      2) Port: 5555, or the port number you configured in your bfagent.conf file.
      3) User name: the z/OS RACF user ID of the builder on the target build machine.
      4) Password and Confirm Password: The builder's z/OS RACF password.
   k. Specify the following values on the Build Command line tab:
      1) Enter this command line into the Command input box. Replace <PDS(MEMBER)> with the data set you created in step 1. Note that the command begins with a leading period.

         ```
         .submitJCL <PDS(MEMBER)>
         ```

      2) Set working directory to a fully qualified USS path on the build machine. This directory will be used as a work directory by the build process. It must exist prior to requesting a build.
   l. Click **Save**.
   m. Request a build.

1) In the Team Artifacts view, select the build definition, right click, and select **Request Build**.
2) Click **Submit**.
3) If a dialog stating that the build engine does not appear to be processing requests is displayed, click **OK** to submit the request.
4) In the Builds view, check the status periodically. Click **Update** to refresh the view.

n. When the build is completed, double-click the build result to view the build log.

Providing JCL through a Rational Build Agent step command

You can also specify JCL inline as part of a Rational Build Agent step command. This job submission method allows you to use substitution parameters to specify values such as the HLQ of the source data sets. The parameters will be replaced with values specified on the Build Definition **properties** tab prior to job submission.

1. Ensure that you have data sets defined that will contain the object decks and load modules that result from COBOL compilation and link editing.
2. Verify that you have defined a RationalBuildAgent build engine. If you have not, follow steps 2a and 2b from the section titled *Submitting JCL Contained in a Build System Data Set*.
3. Create a Build Definition using the Rational Team Concert for System z client.
   a. In the **Team Artifacts** view, right-click the **Builds** icon and select **New Build Definition**.
   b. Click **Next**.
   c. Specify a **Build Definition ID** and select **Rational Build Agent** as the build template.
   d. Click **Next**.
   e. Clear the **Pre Build Command line** checkbox and click **Next**.
   f. Clear the **Post Build Command line** checkbox and click **Finish**.
   g. On the **Overview** tab, select **RationalBuildAgent** as the **Supporting Build Engine**.
   h. On the **Properties** tab, create a new property called `HLQ`. This property will be used throughout the JCL to specify the high level qualifier to be used for source and output data sets on the target build system.
      1) Click the **Add** button.
      2) Select **String** as the property type and click **OK**.
      3) Specify **HLQ** as the name.
      4) Enter the HLQ of the target data sets as the value and click **OK**.
   i. The build agent tab should contain the following values:
      1) **Hostname:** Your build machine's IP address or hostname.
      2) **Port:** 5555, or the port number you configured in your `bfagent.conf` file.
      3) **User name:** the z/OS RACF user ID of the builder on the target build machine.
      4) **Password** and **Confirm Password:** The builder's z/OS RACF password.
   j. j. Specify the following values on the **Build Command line** tab:
      1) Enter this command line into the **Command input** box. Using the option `–c` with the `.submitJCL` command allows you to specify JCL as part of the command. Any occurrence of `${HLQ}` will be replaced with the value specified on the **Properties** tab of the **Build Definition**. Note that the

command begins with a leading period. Be sure to verify that DD statements in the JCL contain values appropriate for your target system.

```
. .submitJCL -c
//HELLO    JOB ,NOTIFY=${HLQ}
/*JOBPARM S=*
// SET HLQ=\'${HLQ}\'
//*
//* COBOL COMPILATION
//*
//COBOL    EXEC PGM=IGYCRCTL,PARM='NODECK,OBJECT,LIB'
//STEPLIB  DD DSN=COBOL.V4R1M0.SIGYCOMP,DISP=SHR
//SYSIN    DD *
       IDENTIFICATION DIVISION.
       PROGRAM-ID. HELLO.
       PROCEDURE DIVISION.
       MAIN.
           DISPLAY 'HELLO, RTCZ.'.
           STOP RUN.
/*
//SYSLIN   DD DSN=&HLQ..SAMPLE.OBJ(HELLO),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT3   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT4   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT5   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT6   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT7   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//*
//* LINKEDIT
//*
//LINKEDIT EXEC
PGM=IEWBLINK,PARM='LIST,LET,MAP,XREF,REUS,RENT'
//SYSLIN   DD *
INCLUDE SYSLIB(HELLO)
NAME HELLO(R)
/*
//SYSLIB   DD DSN=&HLQ..SAMPLE.OBJ,DISP=SHR
//    DD DSN=CEE.SCEELKED,DISP=SHR
//SYSLMOD  DD DSN=&HLQ..SAMPLE.LOAD(HELLO),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//*
```

2) Set the working directory to a fully qualified USS path on the build machine. This directory will be used as a work directory by the build process. It must exist prior to requesting a build.

k. Click **Save**.

l. Request a build:

1) In the **Team Artifacts** view, select the build definition, right-click, and select **Request Build**.

2) Click **Submit**.

3) If a dialog stating that the build engine does not appear to be processing requests is displayed, click **OK** to submit the request.

4) In the **Builds** view, check the status periodically. Click **Update** to refresh the view.

m. When the build is completed, double-click the build result to view the build log.

Scenario 6: Using Antz build extensions to compile a COBOL application

This scenario depends on the successful SMP/E installation of FMIDs HAHB200 (Rational Team Concert for System z Build Toolkit) and HAHD200 (Rational Build Agent).

Apache Ant is a Java based build tool similar to make. Ant allows builders to describe the steps required to build an application using an XML based script. Rational Team Concert for System z provides a set of extensions to Apache Ant called "Antz" that makes it possible to build z/OS based applications using Ant.

This scenario uses Antz to show you how to build a simple "Hello World!" COBOL application on a z/OS build machine. You will learn to use the Rational Team Concert for System z Data Definition, Translator, and Language Definition components to define your build environment. You will then create a shared zComponent project containing your COBOL source code and a build.xml file that defines the Antz build process.

Preparing your Environment

You must prepare your development environment prior to defining builds using Antz. This scenario assumes that you have performed the following tasks:

- Prepared a Jazz Team Server. See Chapter 1, "Installing and configuring Rational Team Concert for System z," on page 1 for information regarding server installation and configuration.
- Created a Project Area
- Created a user to perform the builds. Edit the user and verify that it has the following Client Access Licenses for Rational Team Concert:
  - Contributor
  - Developer
  - Build System
- Installed and configured a Rational Team Concert for System z client. The client should have a repository connection defined for your Jazz Team Server and be connected to your Project Area.
- The Jazz Build System should be installed on the z/OS system that performs builds.
- The Rational Build Agent should be installed and configured on the z/OS system that performs builds.

Notes:

1. The Rational Build Agent will be started with a shell script which is described later in this scenario.
2. This scenario does not depend on the Job Monitor.This scenario does not depend on the Job Monitor.

Configuring the Rational Build Agent shell script

The Jazz Build System contains a sample script named startbfa.sh that can be used to start the Rational Build Agent on the build machine. The script can be found in the *pathPrefix*`/usr/lpp/jazz/buildsystem/buildtoolkit/examples/startbfa`

`directory`, where *pathPrefix* is any prefix specified during the SMP/E installation. This script is required for starting the Rational Build Agent for Beta 2 to specify credentials for the Agent to connect to the Jazz Team Server for System z, as well as to provide access to libraries needed for Antz and native compilation. You can copy this script to a work directory to modify and execute if necessary.

The script must be tailored to suit your environment. Replace the variable strings in the sample script as described in the following table:

Table 3.

Variable	Description
@pathPrefix@	Directory path to prefix the jazz directory. **Note:** This is the prefix to the jazz directory, so your prefix should include any prefix specified as part of the SMP/E installation as well as `/usr/lpp`.
@javaPathPrefix@	Directory path to the IBM 31 bit SDK for z/OS Java 2 Technology Edition V5
@yourUserid@	The jazz user id of the builder
@yourPasswordFile@	The Jazz Password File as defined by the BLZBPASS Job
@stepLib@	STEPLIB DD to use in your z/OS build. For example, `SYS1.LINKLIB:CEE.SCEERUN`
@bfaBinPath@	Directory path to the Rational Build Agent executable directory
@bfaConfPath@	Directory path to the Rational Build Agent configuration file directory

Run the startbfa.sh script when you are ready for the Rational Build Agent to be started and accept build requests.

Configuring the Rational Build Agent Service

By default, the Rational Build Agent Service on the Jazz Team Server checks for build status every 60 seconds. If you'd like to check status more frequently, you can change this default value.

1. Use your web browser to log into the Jazz Build Server at https://>*yourserver*<:9443/jazz.
2. Click **Server**.
3. Under **Configuration**, select **Advanced Properties**.
4. Under the **Build Agent**, change the value of the **Build Agent Loop Task** property from 60 to 10. This causes the Rational Build Agent Service to poll for build status every 10 seconds.
5. Click **Save**.

Create Data Set Definitions

About this task

A data set definition is a new Jazz model object that will be stored in the Rational Team Concert for System z repository. It is a container for information regarding a data set on the z/OS system. All data sets referenced by a build process must have a corresponding Data Set Definition. In this step, you will create the Data Set Definitions required to build a simple "Hello World" application.

Create Data Set Definitions for each of the data sets that will be referenced by the build process. The following table describes the Data Set Definitions that will be required for this scenario.

Table 4. Required Data Set Definitions

Name	Type	Allocation Parameters	Description
COBOL	Destination data set for a zFolder	RECFM(F,B) LRECL(80) SPACE(1,1) CYL	Data set for COBOL source files
LINK	Destination data set for a zFolder	RECFM(F,B) LRECL(80) SPACE(1,1) CYL	Data set for link-edit source files
OBJ	New data set used for build	RECFM(F,B) LRECL(80) SPACE(1,1) CYL	Data set for object decks
LOAD	New data set used for build	RECFM(U) LRECL(0) SPACE(1,1) CYL	Data set for load modules
TEMPFILE	Temporary data set used for build	SPACE(5,5) TRACKS UNIT(SYSALLDA)	Temporary data sets required by the COBOL compiler
IGYCRCTL	Existing data set used for build		COBOL compiler module
SIGYCOMP	Existing data set used for build		Data set of COBOL compiler
IEWBLINK	Existing data set used for build		Link-editor module
SCEELKED	Existing data set used for build		Data set containing the link-edit stubs for C/C++, PL/I, COBOL, and Fortran languages and Language Environment-provided routines

1. Create a data definition that corresponds to the PDS that will contain COBOL source code on the build machine. Select the **Data Set Definition** icon in Team Artifacts view, right click to show the context menu and click **New Data Set Definition**. Name the data definition COBOL.
2. In the General section, specify the following parameters:
   a. Usage: The destination data set for a zFolder This specifies that the new data set definition corresponds to a zFolder contained within a zComponent Project.
   b. Data set Name: COBOL This is the name that will be used on the z/OS system when this data set is created.
3. For data set characteristics, specify the following parameters:
   a. Space Units: Cylinders
   b. Primary Quantity: 1
   c. Secondary Quantity: 1
   d. Directory Blocks: 0
   e. Record Format: FB

f. Record Length: 80
g. Block Size: 0

4. 3. Click **Save**.
5. 4. Repeat steps 1 – 3 to create and save a new data set definition named LINK as described in the table above.
6. Create a data definition named the data definition OBJ.
7. In the General section, specify the following parameters:
   a. Usage: New data set used for build. This specifies that the new Data set Definition refers to an output data set that will be used by the build process. In this scenario, the OBJ data set will be used to hold the object decks produced by the COBOL compiler. If this data set does not exist, it will be allocated during the build process.
   b. Check Prepend prefix to data set name. This indicates that this data set should be prefixed with the high level qualifier associated with the build request.
8. For data set characteristics, specify the following parameters:
   a. Space Units: Cylinders
   b. Primary Quantity: 1
   c. Secondary Quantity: 1
   d. Directory Blocks: 0
   e. Record Format: FB
   f. Record Length: 80
   g. Block Size: 32720 Block size must be larger than zero for beta 2. In the final release, zero will be accepted as a valid value.
9. 6. Create the Data Set Definition for the LOAD library named LOAD.
10. In the General section, specify the following parameters:
    a. Usage: New data set used for build. This specifies that the new data set Definition refers to an output data set that will be used by the build process. In this scenario, the OBJ data set will be used to hold the object decks produced by the COBOL compiler. If this data set does not exist, it will be allocated during the build process.
    b. Check Prepend prefix to data set name. This indicates that this data set should be prefixed with the high level qualifier associated with the build request.
11. For data set characteristics, specify the following parameters:
    a. Space Units: Cylinders
    b. Primary Quantity: 1
    c. Secondary Quantity: 1
    d. Directory Blocks: 0
    e. Record Format: U
    f. Record Length: 80 Note that this value should be 0 for data sets with Record Format U, but the beta 2 release requires a value to be specified in this field.
    g. Block Size: 32720 Block size must be larger than zero for beta 2. In the final release, zero will be accepted as a valid value.
    h. Data Set Type: LIBRARY(PDSE)
12. 6. Create the Data Set Definition named TEMPFILE.
13. In the General section, specify the following parameters:

a. New Temporary data set used for build. This indicates that this data set will be allocated as a temporary file to be used by the build process.

14. For data set characteristics, specify the following parameters:
    a. i. Generic Unit: SYSALLDA
    b. Record Format: U
    c. Block Size: 32720 Block size must be larger than zero for beta 2. In the final release, zero will be accepted as a valid value.
    d. All other fields: Accept the default values.
15. 6. Create the Data Set Definition named SIGYCOMP. This should be the name of the data set on the build machine containing member IGYCRCTL. Unlike the previous Data Set Definitions created, this definition refers to an existing resource on the build system. This definition describes the characteristics of the data set containing the IBM COBOL for z/OS compiler.
16. In the General section, specify the following parameters:
    a. Existing data set used for build. This indicates that this data set will be allocated as a temporary file to be used by the build process.
    b. Member: Blank
    c. Uncheck Prepend prefix to data set name.
17. 6. Create the Data Set Definition named IGYCRCTL. This Data Set Definition describes the characteristics of the IBM COBOL for z/OS compiler.
18. In the General section, specify the following parameters:
    a. Existing data set used for build.
    b. Member: IGYCRCTL
    c. Uncheck Prepend prefix to data set name.
19. 6. Create the Data Set Definition named IEWBLINK . This should be the name of the data set on the build machine containing member IEWBLINK. This Data Set Definition describes the IEWBLINK module that is used to bind a program and store it in a program library.
20. In the General section, specify the following parameters:
    a. Existing data set used for build.
    b. Member: IEWBLINK
    c. Uncheck Prepend prefix to data set name.
21. 6. Create the Data Set Definition named SCEELKED. This should be the name of the data set on the build machine containing the COBOL and LE link edit stubs (generally named CEE.SCEELKED).
22. In the General section, specify the following parameters:
    a. Existing data set used for build.
    b. Member: Blank
    c. Uncheck Prepend prefix to data set name.

Create Translators

About this task

Translators are a new Jazz model object that will be stored in the Rational Team Concert for System z repository. Translators describe an operation to be performed on a file during a build. A set of Translators can be associated with a Language Definition. During a build, the set of Translators associated with a Language Definition will be iterated over and executed for each file associated with the Language Definition.

For this scenario, you will need two Translators; IGYCRCTL and IEWBLINK.

Create IGYCRCTL Translator
About this task

The IGYCRCTL Translator will be used to compile the COBOL program.

1. 1. Expand the **Language Definitions** icon in the Team Artifacts view, select the **Translators** icon, show the context menu and click **New Translator**. Name the translator IGYCRCTL.
2. In the General section, specify the following parameters:
   a. Data Set Definition: IGYCRCTL. This is the Data Set Definition containing the executable module to be used by this translator. Use the Browse... button to select this Data Set Definition from the list of definitions created in previous steps.
   b. Default Options: NODECK,OBJECT,LIB. These options correspond to the PARM parameter field of the JCL EXEC statement.
   c. DD Names List: SYSLIN,,,SYSLIB,SYSIN,ANTPRINT
   d. Maximum Return Code: 0

IGYCRCTL DD allocations table:
About this task

Set up the IGYCRCTL DD allocations table. This table specifies the dataset allocations expected by the module associated with the Translator. In this step, specify the following DD Allocations for use by the COBOL compiler.

1. Click **Add** next to the DD Allocations table. In the dialog, enter the following values:
   a. DD Name: SYSIN. This is the input COBOL source file.
   b. Select **Allocate data set currently being processed (input to translator)**.
2. Add a DD Allocation with the following values:
   a. DD Name: SYSLIN. This allocates the target object module data set.
   b. Select **Allocate data set from a specified data set definition** to indicate that this data set should be allocated using the characteristics specified in a Data Set Definition.
   c. Click **Browse** and select the OBJ Data Set Definition from the list.
   d. Check **Append member name to data set name**. This tells the build process to append the member name of the input data set to the dataset name.
3. Create a TASKLIB DD Allocation with the following parameters:
   a. DD Name: TASKLIB
   b. Check **Allocate data set from a specified data set definition**.
   c. Click **Browse** to select the SIGYCOMP Data Set Definition.
4. d. Create a DD Allocation to be used by Antz to collect build output
   a. DD Name: ANTPRINT
   b. Check **Allocate data set from a specified data set definition**.
   c. Click **Browse** to select the TEMPFILE Data Set Definition.
5. e. Create DD Allocations for the working datasets used by the COBOL compiler during compilation.
   a. Specify SYSUT1 as the DD Name.
   b. Check **Allocate data set from a specified data set definition**.
   c. Click **Browse** to select the TEMPFILE Data Set Definition.

d. iv. Repeat steps a through c to create DD Allocations for SYSUT2, SYSUT3, SYSUT4, SYSUT5, SYSUT6, and SYSUT7.

Create IEWBLINK Translator

About this task

The IEWBLINK Translator will be used to link-edit the "Hello World" module.

1. 1. Expand the **Language Definitions** icon in the Team Artifacts view, select the **Translators** icon, show the context menu and click **New Translator**. Name the translator IEWBLINK.
2. In the General section, specify the following parameters:
   a. Data Set Definition: IEWBLINK.
   b. Default Options: LIST,LET,MAP,XREF,REUS,RENT.
   c. DD Names List: SYSLIN,SYSLMOD,SYSLIB,ANTPRINT
   d. Maximum Return Code: 0
3. Set up the Concatenations Table. This table specifies concatenated datasets that will be used by this Translator.
   a. Click **Add** next to the Concatenations table.
   b. Type SYSLIB in DD Name.
   c. Click **Add**.
   d. Select **Data set definition:** and click **Browse**.
   e. Select **OBJ** and click **OK**.
   f. Click **OK** on the **Add data set definition** panel.
   g. Click **Add**.
   h. Select **SCEELKED** and click **OK**.
   i. Click **OK** on the **Add data set definition** panel.
   j. Click **OK** on the **Add concatenation** panel.

IEWBLINK DD allocations table:
About this task

Set up the IEWBLINK DD allocations table.

1. Click **Add** next to the **DD Allocations** table. In the dialog, enter the following values:
   a. DD Name: SYSLIN
   b. Select **Allocate data set currently being processed (input to translator)**.
   c. Click **OK**.
2. Click **Add** next to the **DD Allocations** table. In the dialog, enter the following values:
   a. DD Name: SYSLMOD .
   b. Select **Allocate data set from a specified data set definition** to indicate that this data set should be allocated using the characteristics specified in a Data Set Definition.
   c. Check **Append member name to data set name**. This tells the build process to append the member name of the input data set to the dataset name.
   d. Click **Browse** and select **LOAD** from the list.
   e. Click **OK**.
3. Click **Add** next to the **DD Allocations** table. In the dialog, enter the following values:

a. DD Name: ANTPRINT
b. Check **Allocate data set from a specified data set definition**.
c. Click **Browse** to select **TEMPFILE** from the list..
d. Click **OK**.

4. Click **Add** next to the **DD Allocations** table. In the dialog, enter the following values:
   a. DD Name: SYSUT1
   b. Check **Allocate data set from a specified data set definition**.
   c. Click **Browse** to select **TEMPFILE** from the list..
   d. Click **OK**.
5. d. Create a DD Allocation to be used by Antz to collect build output
   a. DD Name: ANTPRINT
   b. Check **Allocate data set from a specified data set definition**.
   c. Click **Browse** to select the TEMPFILE Data Set Definition.
   d. Click **OK**.
6. e. Create DD Allocations for the working datasets used by the COBOL compiler during compilation.
   a. Specify SYSUT1 as the DD Name.
   b. Check **Allocate data set from a specified data set definition**.
   c. Click **Browse** to select the TEMPFILE Data Set Definition.
   d. iv. Repeat steps a through c to create DD Allocations for SYSUT2, SYSUT3, SYSUT4, SYSUT5, SYSUT6, and SYSUT7.

Create Language Definitions

About this task

Language Definitions are a new Jazz model object that will be stored in the Rational Team Concert for System z repository. Each file to be built with the Rational Team Concert for System z build process will have a Language Definition associated with it.

For this scenario, you will define a Language Definition for COBOL compilation.

1. Select the **Language Definitions** icon in the Team Artifacts view, right click to show the context menu, and click **New Language Definition**.
2. Specify the following parameters:
   a. Name: COBOL.
   b. Language Code: COBOL.
3. Click **Add** next to **Translators** .
4. Select the **IGYCRCTL** translator.
5. Click **OK**.
6. Click **Save**.
7. Select the **Language Definitions** icon in the Team Artifacts view, right click to show the context menu, and click **New Language Definition**. This Language Definition is for link-editing the program.
8. Specify the following parameters:
   a. Name: LINKEDIT.
9. Click **Add** next to **Translators** .
10. Select the **IEWBLINK** translator.

11. Click **OK**.
12. Click **Save**.

Create a Shared zComponent Project

About this task

Now that the build environment has been defined, you must now create and share a zComponent project. A zComponent Project is a specialized Eclipse project that contains z/OS artifacts (source files, link edit files, etc.) and build metadata used when performing builds on a z/OS system.

In this step, you will create a zComponent project to contain your COBOL source code and share the project with your team.

1. Click **Window > Open Perspective > Other** from the main menu bar.
2. Select **Resource** and click **OK** .
3. Select **File > New > Project** from the main menu bar.
4. Select **zComponent > zComponent Project** and click **Next**.
5. Specify your project name and click **Finish**.
6. Select the created project in Project Explorer, right click to show the context menu and select **Team > Share Project**.
7. Select the default component for the workspace and click **Finish**.

Add System z Build Containers and Associate them with Data Set Definition

About this task

The zComponent Project can be thought of as a collection of z/OS datasets. zFolders are used to represent the datasets in your project. Perform the following to create the zFolders required for this scenario.

1. Expand the project in Project Explorer.
2. 2. Select the zOSsrc folder, right click to show the context menu and click **New > Other > zComponent > zFolder**.
3. Specify **Cobol** as the zFolder name.
4. 4. Select the COBOL Data Set Definition from the drop down selection box for Data Set Definition. This associates the zFolder with a Data Set Definition. When this zFolder is processed by the Antz build, the values specified in the Data Set Definition will be used to allocate the zFolder's corresponding z/OS dataset.
5. Click **Finish**.
6. Follow steps 1 through 5 to create the LINK zFolder and associate it with the LINK Data Set Definition.

Add System z Build Artifacts and Associate them with Language Definitions

About this task

You are now ready to define the artifacts required by this project. You will create a COBOL source file and a file containing the bind instructions for the link editor. You will also associate these resources with Data Set Definitions.

1. Select the COBOL folder, right click to show the context menu and click **New > Other > zComponent > zFolder**.
2. Specify **HELLO.cbl** as the zFile name.
3. Select the COBOL Language Definition from the drop down selection box for Language Definition.
4. Click **Finish**.
5. Type the following in the source editor and click **File > Save**. You can cut and paste from the following example. **Note that the first character should start on the 8th column**. IDENTIFICATION DIVISION. PROGRAM-ID. HELLO. PROCEDURE DIVISION. MAIN. DISPLAY 'Hello, world.'. STOP RUN.
6. Select the **LINK** folder, right click and select **New > Other > zComponent > zFile**.
7. Specify **HELLO.lnk** as the zFile name.
8. Select the LINKEDIT Language Definition from the drop down selection box for Language Definition.
9. Click **Finish**.
10. Type the following in the source editor and click **File > Save**. You can cut and paste from the following example. **Note that the first character should start on the 8th column**. INCLUDE SYSLIB(HELLO) NAME HELLO(R)
11. Select the **HELLO.lnk** icon in the Project Explorer, right click and select **Properties**.
12. Click **zFile Properties** and select the **LINKEDIT** language definition from the Language Definition drop-down list.
13. Click **OK**.
14. Switch to **Work Items** perspective.
15. Open the **Pending Changes** view. Check in and deliver all changes.

Create a New Build Definition

About this task

Rational Team Concert for System z provides a new Build Definition template named "Antz – Rational Build Agent". This template allows you to define an Ant based build for use on System z using the Rational Build Agent.

1. In the Team Artifacts view, right click on the **Build Engines** icon and select **New Build Engine**.
2. Specify the Build engine ID as **RationalBuildAgent**, choose a Project or Team Area, and click **Save**.
3. In the Team Artifacts view, right click on the **Builds** icon and select **New Build Definition**.
4. Click **Next**.
5. Specify a Build Definition ID and select the **Antz - Rational Build Agent** template.
6. Click **Next**.
7. Check **File Agent Jazz Source Control**. This specifies that you want to use the Rational Team Concert for System z File Agent to extract the files from the repository and place them on the target build system.
8. Click **Next**.
9. Check **Job Output Publishing**. This option enables build output reporting for this Build Definition.

10. Click **Finish**.

Build Definition Editor
About this task

In the Build Definition Editor, do the following to configure this Build Definition:

1. On the Overview tab, check **RationalBuildAgent** for the Supported Build Engines value.
2. The Build Agent tab should contain the following values:
   a. Hostname: Your build machine's IP address or hostname
   b. Port: 5555 or the port number you configured in your bfagent.conf file
   c. User name: The z/OS RACF user ID of the builder on the target build machine.
   d. Password and Confirm Password: The builder's z/OS RACF password
3. When the Build Agent tab has been configured, click **Test Connection** to test the connection to your build agent. If the connection was successful, you will see the results similar to the following: Message: Socket created successfully. Authentication Pass. Platform: os/390 19.00 03 Version: 7.1.0.0-0-0241 PingResult: PingOk ExitStatus: 0
4. On the Job Output Publishing tab, ensure that **Publish job output logs** is selected.
5. On the **File Agent SCM Configuration**, set the following values:
   a. Build Workspace: Click **Select** to select the repository workspace to be built.
   b. Load directory: Specify an absolute path in USS to be used to store non-MVS build artifacts such as the Antz build.xml file. The user id associated with the builder must have read/write/execute permissions on this directory.
   c. Dataset prefix: Specify the prefix to be prepended to managed data sets. For example, if "BUILDER" was specified as the dataset prefix, the dataset BUILDER.COBOL would be allocated for the COBOL zFolder associated with the COBOL Data Set Definition in previous steps. Refer to Chapter 3, "Known restrictions," on page 69 for information on a restriction and workaround for the prefix.
6. The Build File field on the Antz tab should be set to the absolute USS path of the Antz build file. If you want to use the build.xml file contained in the zComponent project you created previously, the path will be <load directory>/<project>/build.xml. If you named the zComponent project DEMO, the path to the build file could be given as ${teamz.scm.fetchDestination}/DEMO/build.xml. Note that ${teamz.scm.fetchDestination} is a property whose value points to the directory specified in the Load directory field of the File Agent SCM Configuration tab.
7. Click **Save**.

Request a Build
About this task

You have now successfully defined all of the pieces needed to complete this scenario. To request a build, perform the following steps:

1. In the **Team Artifacts** view, select your build definition, right click, and select **Request Build**.
2. Click **Submit**.

3. If a warning dialog stating that the build engine does not appear to be processing requests is displayed, click **OK** to submit the request.
4. 4. In the **Builds** view, check the status of your build periodically. Click **Update** to refresh the status view.
5. When the build is completed, double-click the build result.
6. Go to the **Logs** tab.
7. Double-click the log file. The Logs tab should contain the following files:
   a. ANTPRINT1.log This file contains the output from the COBOL compiler.
   b. ANTPRINT2.log This file contains the output from the link-editor.
   c. build-XXXXXXXXXXXXX.log This file contains the Build Forge Agent Service build output.
8. Look at the last lines of the log. If you see a **Status:OK** message, you have successfully completed this scenario.
9. The **Downloads** tab of the build results should contain the following files:
   a. build.properties A properties file containing all properties and their values used for the execution of this Antz build.
   b. buildableFiles.xml An XML file containing information about each file that was processed by the Antz build. Only those files associated with a Language Definition are included in this list.
   c. fetchedFiles.xml An XML file containing information about each file that was extracted from the repository during this build.
   d. macrodefs.xml An XML file containing Ant macros that were generated for use by the build. The contents of these macros are created based on values specified in Language Definitions associated with files processed by this build.

Chapter 3. Known restrictions

This chapter documents restrictions functions that cause problems in Beta 2.

Failure on load of an empty component or empty component with blanks in workspace name

To work around this issue, ensure that any workspace being used during beta testing does not contain any empty components or components that have a blank in the component name.

File Agent allocates incorrect file when prefix is lower case

To work around this issue, ensure that the data set prefix is specified in your build definition using all upper case characters. For beta 2, we only support a prefix with a single segment. A prefix that contains more than one segment causes an error. For example, a prefix of ″GSLADE″ is acceptable, but a prefix of ″GSLADE.DEV″ causes an error.

Appendix A. Running the Jazz Build Engine on z/OS

In addition to the Rational Build Agent, you can also run the Jazz Build Engine with the Rational Team Concert for System z beta on z/OS, Windows, or Linux.

Setting up the Jazz Build Engine is optional. Complete these steps if you decide to run the Jazz Build Engine on z/OS. These steps are only required if you decide to run builds on z/OS.

Before you perform these setup steps, review "Lesson 8: Team Builds" in the getting started tutorial listed in *Scenario 1: Getting started with Rational Team Concert for System z*.

You must define a build engine instance using the Rational Team Concert for System z client. Follow the instructions in the tutorial to create a build engine.

Before you start a build engine, create an encrypted password file for the build engine to use to prevent casual observation:

1. Tailor member BLZBPASS, in *hlq*.SBLZSAMP, where *hlq* is the high level qualifier used during SMP/E installation. This password file can also be used in your build scripts.
2. Submit the modified JCL.
3. The job must end with a return code 0.

   Important: Do not save the modified JCL with any password in it.

To run the Jazz Build Engine:

1. Tailor member BLZBENG, in hlq.SBLZSAMP, following the instructions contained in the sample JCL.
2. Submit the modified JCL. The job must remain active and the following messages must end the STDOUT:

   ```
   2009-03-27 11:08:46 Running build loop...
   2009-03-27 11:08:46 Waiting for request..
   ```

When you submit a build request to this build engine from Rational Team Concert for System z, you will see a message similar to:

```
2009-03-27 20:00:38 Found a request for build definition "SampleBuild"
```

Note: Make sure that the user ID that is associated with the build engine has read, write, and execute permissions to the directory where you installed the Build System Toolkit for System z, as well as appropriate access to any directories or artifacts associated with your build scripts or build definitions.

Appendix B. Troubleshooting the Jazz Team Server

You can use the administrative Web interface to troubleshoot server issues.

Before you begin

You must be logged in to the Administrative Web user interface and be a member of the JazzAdmins group.

About this task

To troubleshoot server issues, in the from the administrative Web user interface, click the **Server** tab. On this page, you can find server state information that can help you troubleshoot problems.

Tip: Go to jazz.net Tech Notes at https://jazz.net/learn/tech-notes/tech-notes.jsp or Server troubleshooting FAQ at https://jazz.net/wiki/bin/view/Main/ServerTroubleshootingFAQ to find additional troubleshooting tips.

The table below lists common problems and ways to troubleshoot them.

Problem	What to check
Database is not available	Check the server status: click **Status summary**. • On the Status Summary page, in the Server Status pane, verify that the database status is connected. If the status is unavailable, check the Database Status pane for details. Possible failures include a database configuration that is not valid and a valid configuration to a database that is not initialized. **Note:** The Derby database supports only one connection; if another server instance is running, it might be using the only database connection. If this problem occurs, the Service Error Summary pane lists multiple errors. • In the Server VM pane, verify that the server is running the correct JDK. • In the Service Error Summary pane, check for error messages.

Problem	What to check
Unable to change configuration properties	Check the server configuration properties: under Configuration, click one of the following items: • **E-mail Settings** • **Database Connections** • **Feed Settings** • **License Key Management** • **Advanced Properties** On these pages, you can view and update configuration properties. When you save changes to configuration properties, they are propagated to the teamserver.properties file. If the database is not connected, you can change configuration properties only by editing the teamserver.properties file.
Failing services	Check the status of services: click **Component Status**. On the Component Status page, check the stack trace for more information about failing services.
Slow server activity	Check the running services: click **Active Services**. On the Active Services page, check the running services and their stack traces. Check for services that run for an extended period of time. Check the server activity: click **Statistics**. On the Server Statistics page, check the server activity, such as statistics for Web services, asynchronous tasks, and cache behavior.

Additional resources and tips

For more information, you can access the log feed:

- If you are running a secure connection, access the log feed from this location: https://localhost:9443/jazz/service/com.ibm.team.repository.common.internal.IFeedService?category=SystemLog
- If you are not running a secure connection, access the log feed from this location: http://localhost:9080/jazz/service/com.ibm.team.repository.common.internal.IFeedService?category=SystemLog

If the log feed is not available, view the Tomcat log file from a Windows console window or at *JazzInstallDir*`/tomcat/logs/catalina.out`, where *JazzInstallDir* is the location where you installed the server.

If the teamserver.properties file is not located during startup, the server does not work properly and the Tomcat log and log feed contain errors. The server.startup script provides the path to the property file.

Ensure that the default Tomcat connection ports 9080 and 9443 are not in use. The connection ports are defined in the *JazzInstallDir*/tomcat/conf/server.xml file, where *JazzInstallDir* is the location where you installed the server.

If the problems persist, consider reinstalling the database or the Web archive (WAR) file. You can reinstall the database from the original distribution file (repositoryDB.zip) or from a backup file.

Troubleshooting the Jazz Team Server setup wizard

There are several actions you can take if the server setup wizard does not load.

About this task

Check the following items:

1. Use the following URL to verify that the application server has started: `http://localhost:9080`.
2. Verify the Jazz Team Server has started by logging in to the Jazz Team Server Administrative Web user interface using this URL: `https://localhost:9443/jazz/admin`. If the page does not load or the server has errors, the server did not start correctly. See the troubleshooting information in the *Troubleshooting server issues* section of this guide.
3. The URI root for the Jazz Team Server path should be `/jazz`. For example `https://example.com:9443/jazz` must be used, rather than `https://example.com:9443`.

Appendix C. Troubleshooting Rational Build Agent issues

This section describes procedures that you can use to troubleshoot the Rational Build Agent installation process and other issues.

Troubleshooting the Rational Build Agent installation on z/OS

You might receive error messages after building the agent source code on z/OS. This topic describes fixes for some common errors.

The configure-zos script sets some common values and performs some basic checks to identify the headers and functions available on your system.

Because of variations in z/OS system configurations, the ./configure-zos script might run without errors but you might see the following errors when you run the ./build-zos script.

CEE3501S The module CCNDRVR was not found.

FSUM3066 The COMPILE step ended with the following return code:

```
-1: EDC5083I An error  occurred attempting to load a module into
storage.
```

This error indicates that a required dynamic library cannot be loaded by the compiler.

Run the command: `% export STEPLIB="SYS1.SCCNCMP"`
Rerun the ./build-zos command. If the command fails again, contact your system administrator for assistance in locating the required library.

IKJ56228I DATA SET CEE.SCEEOBJ NOT IN CATALOG OR CATALOG CAN NOT BE ACCESSED

FSUM3066 The COMPILE step ended with the following return code:

FSUM3052 The data definition name C8961 cannot be resolved. The data set was not found. Ensure that data set name CEE.SCEEOBJ is specified correctly.

This error indicates that the linker was unable to locate a system library that it needs to complete the compilation. Run the commands:

```
% export _C89_LSYSLIB=SYS1.SCEELKED:SYS1.SCEELKEX
```

```
% export _C89_PSYSLIB=SYS1.SCEEOBJ
```

Rerun the ./build-zos command. If the command fails again, contact your system administrator for assistance in locating the required libraries.

IEW2456E 9207 SYMBOL xxx UNRESOLVED

The unresolved symbol errors indicate that the build expected a symbol to be defined by your system C library that is not actually there. In most cases, this is a symbol that is often missing from other systems too, and there will be a setting in config.h to work around the problem.

For example, your system might not define the unsetenv function. The configure-zos script should normally detect this; if it does not, edit the config.h file that is provided with the agent source pack, as follows:

Change `#define HAVE_UNSETENV 1` to `#undef HAVE_UNSETENV`.

Rerun the ./build-zos command to correct the problem.

Note: Similar #define statements exist for other functions.

Troubleshooting the Rational Build Agent

This section describes procedures that you can use to troubleshoot the Rational Build Agent that is not working correctly. Go through the procedures in order.

Testing host name resolution

Verify that the agent can be reached from the machine that is running the Jazz Team Server for System z.

Ping your host system, for example:

```
C:\>ping mvs269.rtp.raleigh.ibm.com

Pinging mvs269.rtp.raleigh.ibm.com [9.12.128.138] with 32 bytes of data:

Reply from 9.12.128.138: bytes=32 time=52ms TTL=55
Reply from 9.12.128.138: bytes=32 time=52ms TTL=55
Reply from 9.12.128.138: bytes=32 time=52ms TTL=55
Reply from 9.12.128.138: bytes=32 time=51ms TTL=55

Ping statistics for 9.12.128.138:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 51ms, Maximum =  52ms, Average =  51ms

C:\>
```

A message similar to the following message indicates a problem:

```
Unknown host
```

There is a problem with the network configuration of the of the Jazz Team Server for System z. Contact your network administrator.

Testing the connection

You can test the connection to the agent by using telnet.

To test the connection from the command line:

1. Connect to the agent with a telnet command. If you are logged on to the host where the agent is running, you can use localhost as the host name.

   ```
   telnet hostname 5555
   ```

   This response indicates a successful connection:

   ```
   200 HELLO - BuildForge Agent v7.0.1.buildnumber
   ```

2. Check authentication by issuing the following commands, using your login credentials:

   ```
   telnet localhost 5555
   username user name
   password  password
   cmd ping
   go
   ```

   Note: The commands are not processed until you type the `go` command.

The following message indicates success:

```
AUTH: set user account to <user name>
```

The following output of a telnet session is typical. In particular look for `RESULT 0` at the end of the output as an indication of success.

```
200 HELLO - BuildForge Agent v7.1.0.0-0-0241
username bgreen
password rtc4fun
cmd ping
go
320 AUTH AuthOk["bgreen"]
320 SET EnvSet["BF_AGENT_VERSION","7.1.0.0-0-0241"]
320 SET EnvSet["BF_AGENT_PLATFORM","os/390 19.00 03"]
320 EXEC Locale["C"]
320 PTY Pty[*PtyPipe]
320 EXEC Locale["C"]
320 EXEC ExecShellPath["/bin/sh"]
300 HEARTBEAT 1
310 PLAT os/390 19.00 03
320 PING PingOk
251 RESULT 0
260 EOR
```

Additional troubleshooting procedures

To troubleshoot a Build Forge agent, try these procedures:

- Run **bfagent** from a shell. The correct response is similar to this message:

  ```
  200 HELLO - Build Forge Agent v7.0.1.122
  ```

  If you receive a message similar to the example and there are shared library problems, you will receive messages regarding those problems. You can resolve most shared library problems by setting the path correctly.
- Check that the agent is listening. Use the following command (assuming that the port is the default, 5555):

  ```
  telnet localhost 5555
  ```

  A 200 HELLO response indicates that the agent is listening. If you do not get this response, check your systems network configuration. Verify that the **inetd** configuration is correct, or check with your system administrator.
- Check authentication. Issue the following commands, using your login credentials:

  ```
  telnet localhost 5555
  username <user name>
  password <password>
  cmd ping
  go
  ```

  A message similar to the following message indicates that the authentication is working correctly:

  ```
  AUTH: set user account to <user name>
  ```

bfagent Reference

The Build Forge agent is normally run as an auto-starting service or daemon, but for the Rational Team Concert for System z beta, you can start the bfagent program from the command line.

The syntax for the command is:

```
bfagent [-f configfile] [-s]
```

Options

-f configfile
Run using the configuration file in configfile, rather than `bfagent.conf`. This is a runtime option on z/OS.

-s Start as a standalone service. This option is an alternative to starting bfagent with either the **inetd** or **xinetd**.

bfagent.conf Reference

The bfagent.conf file stores settings for how the Rational Build Agent runs. The file is in the same directory as the bfagent executable file.

Note: For Rational Team Concert for System z V2 Beta 1, this section provides reference to Rational Build Agent configuration for multiple platforms and some statements might not apply to z/OS.

The file lists all settings and internal defaults. Inactive settings are commented out.

Settings

activity_log *path*
Turns on activity logging. The information is appended to the file specified by *path*. The path must exist and the agent user must have write permission for it.

Note: The agent does not report an error if the path does not exist or if it cannot write to the file.

Important: There is no limit on file size. The file must be manually deleted. This setting is intended to be used temporarily for debugging the agent. It is not intended as a permanent log for a working agent.

allow IP-address-or-range [...]

Use this setting for these conditions only:

- Agents running on Windows
- Agents running in standalone mode on UNIX or Linux when the -s option on startup is used.

This setting limits connections to the agent. Connections are allowed only from the IP addresses that match IP-address-or-range. By default, connections are allowed from all addresses.

Specify one or both of the items:

- **IP Address**: A fully-qualified IPv4 or IPv6 address. For example, for IPv4, 255.192.192.003. The specific IP address is allowed.

- **IP Address range**: A partially-qualified IPv4 or IPv6 address. These examples are correct for IPv4, 192.168 or 192.168.63. All IP addresses that match this qualification are allowed.

Note: If you are running the agent on a superserver such as **inetd** or **xinetd**, use another method to control access. You can use a firewall, TCP wrappers (hosts.allow and hosts.deny), or the built-in filtering capability of **xinetd**.

bind This setting allows you to specify an explicit bind address for the agent. This, together with the ″port″ setting, determines how the agent will listen for connections when it is started with the -s command line option. The value given in the bfagent.conf file will force the agent to bind to the IPv4 localhost address; thus, the agent will only receive connections from a console that is on the same machine. Example: `bind 255.192.192.003`

Note: It has no effect on agents that are started by the system's service architecture, such as **inetd**, **xinetd**, or **launchd**.

ccviewroot root-path
This setting specifies the default view root for this host. See ClearCase documentation on init for more information. The internal defaults are as follows:

- ccviewroot /view

command_output_cache size
This setting causes the agent to cache output until it reaches the specified size in bytes. The internal default is not to cache. Using a cache can significantly improve agent performance and reduce network overhead. The cache size depends on how much output that commands wproduce.

Minimum value: 2048. A value of 2048 is used internally if the setting is less than that.

cygwin

This setting is used only with agents on Windows.

This setting enables the agent to work on a Windows host using cygwin, a Linux-like environment. When using cygwin, a number of Linux tools are available to the agent.

When you use this setting, you might need to set cygwin_script_magic and shell settings also. The example shows one way to configure these settings:

```
cygwin
shell C:\cygwin\bin\bash.exe --login -c "%s"
cygwin_script_magic #!/bin/bash
```

The shell setting must match your installation of Cygwin.

cygwin_script_magic

This setting is used only with agents on Windows when **cygwin** is set.

This setting specifies the `#!` line to use when executing steps. The default is `#!/bin/bash`.

default_logon_domain
Specifies the domain to use when an authentication request does not include a domain. If not specified, the agent machine's domain is used.

disable_telnet
For best results, use telnet to test the agent connection.

For the agent, there is some built-in processing overhead associated with processing and correctly handling telnet control sequences.

Use this setting to disable the agent from handling special telnet character codes that can slightly improve performance. In product environments, use this setting to benefit from the improved performance.

disable_transcode
Turns off processing that the agent performs to convert international data when the operating system is not using UTF-8 encoding. To avoid mixed encodings and data corruption, use UTF-8 for the agent operating system.

If the operating system does not use UTF-8 encoding, the agent must convert data to the correct encoding for the operating system's locale settings.

If your operating system does not use UTF-8, use this setting for best results and improved performance of the agent.

enable_agent_dll
This setting enables DLL process tracing, which is a debugging tool.

env_recursion_limit number-of-recursions
Sets the variable-replacement recursion limit for pre-parsing. If not set, the limit is 32.

extensions
This setting specifies paths to external libraries of functions. The functions can be used as dot commands in a step. If this setting is not specified, external libraries are not loaded.

During parsing, the first token in the step command is taken as the function name. The second token is a string, and the third is an integer timeout value (in seconds).

Requirement: Dynamic loader support in the operating system. For example, in UNIX or Linux you need /usr/include/dlfcn.h. These default values are used internally.

- UNIX or Linux: /usr/local/bin/bfextensions.so
- Windows: c:\program files\ibm\build forge\agent\bfextensions.dll

getaddrinfo_using_addrconfig
This setting is used only for running the agent as a standalone service on UNIX or Linux operating systems (bfagent -s). This setting makes the agent use AI_ADDRCONFIG when calling getaddrinfo() to select a listening interface. By default, AI_ADDRCONFIG is not used.

If you use this setting, the agent ignores interfaces that do not have a properly configured address. It listens only for interfaces that have a properly configured address.

lang *lang-code*

Use this setting only when the Management Console does not provide a valid language.

This setting specifies the language that the agent uses to write messages and command output. Typically it is not set explicitly because the agent uses the language that the Management Console specifies. However, setting the language can be useful if the desired locale is not available on

the computer. The setting is also useful as a backup, in case the Management Console fails to communicate a language or communicates an invalid language.

The internal default is `en`, as if it were explicitly set as follows:

```
lang en
```

leave_tmp_file

Use this setting only while you are troubleshooting.

This setting causes the temporary file that is used to hold step commands to be retained, rather than deleted after command execution. In troubleshooting, the file can be compared to the steps as they are displayed in the Management Console.

Note: Do not use this setting for typical operations.

locale locale-code.charset-code

This setting is used only with UNIX and Linux operating systems. Windows handles locales differently.

This setting specifies the language and multibyte character set that localized applications use. This setting works by setting the LANG environment variable for the agent context.

To set up the agent to treat command output as US English UTF-8, use the UTF-8 locale for your operating system. For example, on Linux use the following representation.

```
locale en_US.UTF-8
```

To determine the correct representation of the UTF-8 locale for your operating system, run the **locale -a** command.

If this setting is not specified, the agent uses the locale of the operating system. This setting is a convenience. This setting is especially useful if the default locale of the operating system is not the locale that you want the agent to use. The setting is especially useful if changing the system locale to meet agent requirements is not practical.

magic_login user:encoded-password

The agent typically uses administrative privileges such as root or admin to log on to the operating system. The magic_login setting is an alternative to standard system authentication. With this setting, the system can authenticate your login with a single user name and password.

If the agent is run as the root or admin user, this setting is ignored and normal authentication is attempted.

The agent runs all commands using the permissions of the user who started the agent, not the user name used to log in.

This setting is used in only these situations:

- When running the agent with administrative privileges is not possible. For example, use this setting with UNIX systems that do not work with PAM.
- When running the agent with administrative privileges is not permissible because of security policies.

To configure a login for the agent:

1. To create a server authentication that uses a user name and password. In the Management Console, click **Servers** → **Server Auth**.
2. For this example the user name is `build` and the password is `MySecretPassword.`
3. Create a server that uses the agent. Associate the server authentication with this server in the **Authentication** field.
4. Generate an encoded password for the agent. In the installation directory for the agent, run **bfagent -P** with the password that you choose.

   An SMD5 hash-encoded password is returned, as follows:

   ```
   bfagent -P "MySecretPassword"
   eca0b7f2f4fbf110f7df570c70df844e1658744a4871934a
   ```
5. In `bfagent.conf`, set magic_login to use the desired user name and encoded password.

   ```
   magic_login build:eca0b7f2f4fbf110f7df570c70df844e1658744a4871934a
   ```
6. Start the agent.
7. Test the server connection. In **Servers**, select the server, and then click **Test Server**.

map *drive-and-user-spec***[; ...]**
This setting specifies a mapped drive. Some systems might require drive mappings. For example, a drive mapping might be required because a shell is run from a shared drive. Mappings specified on the agent are performed before mappings specified by _MAP environment variables in the Management Console. This example illustrates two drive mappings:

```
map X:=//host1/share;Z:=//host2/share(username,password)
```

map_drive_is_failure
When specified, this setting causes a step to fail upon encountering an unmapped drive specification, before step execution. If this setting is not specified, steps ignore drive failures and attempt to run the step. In that case ensure that the failure generates a meaningful error message.

no_preparse_command
This setting disables the variable-expansion parsing that the agent typically performs on a command before passing the command to the shell. See also the _NO_PREPARSE_COMMAND environment variable, which can be used for an single project or step.

no_pty

This setting is used only with agents that are running on UNIX or Linux systems.

This setting can be used to help prevent the system shell from locking up when the shell interacts with the pseudoterminal of the agent. This setting is typically used with HP/UX and z/OS. You can also use two other methods to help prevent this kind of lockup:

- Use an alternate shell.
- Use the nologonshell setting

The **no_pty** setting disables the pseudoterminal allocation.

Note: Using **no_pty** affects some commands. For example, the **ls** command returns output in a single column rather than three columns. If you use this setting, test thoroughly before you deploy the job to a production environment.

nologonshell

Use this setting only with agents that are running on UNIX or Linux.

This setting causes the shell that the agent runs to be a normal shell, not a logon shell. This setting is often in these cases:

- The logon shells provide verbose output.
- The logon shells change environment settings in unwanted ways.
- The logon shells attempt to communicate interactively with the user.

When set, standard methods of requesting that the shell be a normal shell rather than a logon shell are used. This may not work on all platforms and in such cases, the shellflag setting may be used to pass flags to the shell in order to modify its behavior.

Those behaviors are not desirable for the agent, because it runs as a user without being an interactive user.

Note: The Mac OS X 10.5 system uses /bin/bash, which does not respond to nologonshell. Use `shellflag -l`.

Note: The z/OS operating system always uses the /etc/profile script for both logon shells and non-logon shells. You might need to change the contents of the script or use another shell if its behavior does not work well with the agent.

See also the **shellflag** setting. Flags can be used to change logon script behavior.

password_encrypt_module *dll_path*;*conf_path*

Required to enable SSL on the agent. It specifies paths to a DLL and configuration file.

- *dll_path* is the path to `bfcrypt.dll` (it is typically `./bfcyrpt.dll`).
- *conf_path* is the path to `bfpwcrypt.conf` (it is typically `./bfcrypt.conf`).

port *port-number-or-range* **[...]**

This setting is used only with agents that are running in standalone mode on UNIX or Linux when you issue the **-s** option on startup.

This setting specifies the port that the agent uses to listen for connections with the Management Console.

- Agents running in standalone mode on UNIX or Linux (-s option on startup).

Specifies the port that the agent uses to listen for connections with the Management Console.

Note: The port is set to 5555 by default. For UNIX or Linux the setting is in `/etc/services`.

shell *shell_name* **[***options***]**

This setting specifies the default shell. Internal defaults are as follows:

- Windows: `shell cmd.exe /q /c "%s"` unless the following settings are used:
  - If the `cygwin` setting is used, the default is `shell C:\cygwin\bin\bash.exe --login -c "%s"`
  - If the `cygwin` setting is not used, the default is `shell cmd.exe /u /q /c "%s"`

- UNIX or Linux: The shell set for the user account, or `/bin/sh` if the user's shell can not be determined. Note that you cannot specify parameters in this setting, but you can use the `shellflag` setting to pass them. The agent automatically forces the default to be a logon shell by inserting a hyphen. For example, `/bin/ksh` is sent as `-ksh`. If `shell` is set explicitly, then `nologonshell` is set implicitly. See `nologonshell`.
- *System i*: Set the shell value to /bin/sh

You can override this setting from within a step. A step that starts with a line containing `#!` overrides the shell setting and the `nologonshell` setting is used to run the step commands.

shell_compatible_undef_vars
This setting forces the representation of undefined variables to be an empty string. If not set, the representation is the variable name for variables of format $VAR, ${VAR}, or %VAR% and the empty string for $[VAR].

shellarg

This setting is used only with agents that are running on UNIX or Linux.

Use this setting if it seems that commands are being scrambled. Some shells on Red Hat Linux Enterprise require this setting.

The setting changes the way a command script is passed to the shell. Normally the script is passed through standard input:

```
/bin/sh < /tmp/bfshellscript.sh
```

This setting causes scripts to be run by passing them as parameters:

```
/bin/sh /tmp/bfshellscript.sh
```

shellflag *flag*

This setting is used only with agents that are running on UNIX or Linux.

This setting adds a flag when a shell is running. Only one flag can be specified. It is typically used to disable **rc** script processing in order to reduce output or undesired processing.Examples:

- csh and derivatives: use `shellflag -f` to disable rc script processing.
- bash: use `shellflag –-noprofile` to disable profile script processing.

ssl_ca_location *path*
Specifies the keystore file that contains the certificate authority. If the agent runs as a service, use an absolute path.

ssl_cert_location *path*
Specifies the keystore that contains the private certificate. If the agent runs as a service, use an absolute path.

ssl_client_authentication [true | false]
Set to true to require client authentication when a connection is made to the agent. If true, the Build Forge engine's certificate must be added to the agent's certificate authority keystore.

ssl_cipher_group [*grouplist* **| ALL]**
Specifies individual cipher groups to use. Can be set to ALL.

ssl_cipher_override *cyphers*
Overrides the cipher group. Specify the ciphers to use.

ssl_key_location *path*
Specifies the keystore file that contains the key. If the agent runs as a service, use an absolute path.

ssl_key_password *password*
Password for the key. This property is stored in clear text by default. You can enable the agent to encrypt this password using its own key or the Build Forge server's key.

ssl_protocol *protocol*
The SSL handshake protocol to use, one of SSL, SSLv2, SSLv3, SSL_TLS, TLS, TLSv1. The protocol must match the protocol used by the Build Forge server.

update_path *path*

This setting identifies the full path to the Rational Build Agent executable. The setting is established automatically during installation. The directory is a default directory for the operating system or the installation directory that you specify.

Note: This setting is ignored on Windows agents. The update path is taken from registry keys. The keys are set during agent installation.

win_reexec_after_auth
Add this setting if you need to run agent commands on a network share file system using Build Forge server authentication credentials. For example, to modify files in a ClearCase dynamic view, the agent must access ClearCase files on a networked shared file system.

The Rational Build Agent initially starts up with Windows system account credentials. To run commands, the agent later authenticates with Windows using Build Forge server authentication credentials.

Without this setting, the network share recognizes only the initial Windows system account credentials and ignores the subsequent server authentication credentials that are needed to access and write to files on the network share file system.

The win_reexec_after_auth starts a new process after authenticating with Windows again by using the server authentication credentials and forces the shared file system to recognize the changed credentials.

When you use the win_reexec_after_auth setting, the agent runs as a service and does not distinguish between commands that access network share files and those that do not, so you might notice a performance impact.

Appendix D. Additional Jazz Team Server for System z repository tools tasks on z/OS

Before completing any additional tasks, review the following Rational Team Concert topic: Repository tools overview.

These instructions assume that you are using the default properties files of `teamserver_derby.properties` (for Derby) or `teamserver_db2z.properties` (for DB2 9.1 on z/OS).

Exporting the database using the Jazz Team Server repository tools

This topic describes the steps and requirements for exporting the database using repository tools.

About this task

To export the database with the Jazz Team Server for System z repository tools, you must:

1. Configure one of the following members using the instructions in the sample JCL:
   - If you are using DB2 for z/OS, configure member BLZEXPOR in *hlq.SBLZSAMP*.
   - If you are using Derby, configure member BLZEXPO2 in *hlq.SBLZSAMP*.
2. Submit the modified JCL and check the job log. The following message must end the STDOUT: `Export completed successfully. The user "ADMIN" has logged out of the database.`

   Notes:

   a. All the server processes that are connected to the repository database must be stopped, including the Jazz server and the Jazz build engine. If processes are not stopped, the export will fail.

   b. When you use repository tools, you can specify the codepage to use by specifying the JVM option –Dfile.encoding.
      - If you want to use UTF-8, specify -Dfile.encoding=UTF-8.
      - If you want to specify Cp1047, which is an EBCDIC encoding used for C/Java programming, specify -Dfile.encoding=Cp1047.
      - If you export in UTF-8, you can move your .tar file to a distributed system and import it using that system.
      - If you export in Cp1047, your .tar file can only be imported using a repository tools import performed on System z.

Running repository tools to import a Jazz Team Server for System z database

This topic describes the steps and requirements for importing a Jazz Team Server for System z database using repository tools.

About this task

To import a Jazz Team Server for System z database by running repository tools, you must:

1. Configure one of the following members using the instructions in the sample JCL:
   - If you are using DB2 for z/OS, configure member BLZIMPOR in *hlq.SBLZSAMP*.
   - If you are using Derby, configure member BLZOMPO2 in *hlq.SBLZSAMP*.
2. Submit the modified JCL and check the job log. The following message must end the STDOUT: `Import completed successfully. The user "ADMIN" has logged out of the database`.

   Notes:

   a. All the server processes that are connected to the repository database must be stopped, including the Jazz server and the Jazz build engine. If processes are not stopped, the import will fail.

   b. Pay attention to the JVM properties, like **Xms** and **Xmx**.

Creating database tables using Jazz Team Server for System z repository tools

If you need to recreate a new Jazz Team Server for System z repository, you can use the Jazz Team Server for System z repository tools to create the required database tables and indexes for the Jazz Team Server for System z.

Before you begin

Ensure the `teamserver_derby.properties` file or the `teamserver_db2z.properties` file have been configured correctly as described in earlier in this guide and, if you are using DB2 for z/OS, have previously created the DB2 database.

About this task

Then you must:

1. Configure one of the following members by following the instructions in the sample JCL.
   - If you are using DB2 for z/OS, configure member `BLZCREDB` in *`hlq`*`.SBLZSAMP`.
   - If you are using Derby, configure member `BLZCRED2` in *`hlq`*`.SBLZSAMP`.
2. The Jazz Team Server for System z and the Jazz Team Server repository tools require access to the DB2 V9 JDBC license jar file on your system. This file is named `db2jcc_license_cisuz.jar` and is located by default at `/usr/lpp/db2910_jdbc/classes`. If the file is in a different directory, then modify the sample job to point to the correct location.
3. Submit the modified JCL and check the job log. The following message must end the STDOUT: `The database tables were created successfully.` The details are in the JCL sample.

Appendix E. Antz Tasks and Datatypes

Antz provides a number of custom data types and tasks that allow the build script author to define and select a set of buildable files from within a repository workspace. Additional custom tasks allow execution of the translator steps defined in a Language Definition through the invocation of generated Ant macro definitions. For general information regarding getting started with Apache Ant, please see http://ant.apache.org. Antz custom data types that can be included in a build script (build.xml) are:

Table 5. Antz custom tasks and data types

Class name	Tag	Description
BuildableResourceCollection	<antz:buildableset>	An Ant ResourceCollection that represents a set of buildable resources.
ComponentSelector	<antz:componentselector>	A custom resource selector that allows the resources to be included in a BuildableResourceCollection based on the name of the component that defines the resource.
ProjectSelector	<antz:projectselector>	A custom resource selector that allows the resources to be included in a BuildableResourceCollection based on the name of the project that defines the resource.

Antz provides the following custom task for use in z/OS based build scripts:

Table 6. Custom task for z/OS based build

Class name	Tag	Description
Compile	<antz:compile>	An Ant Task that compiles a BuildableResourceCollection that is provided as its child element.

The following task and data types are also provided by Antz. However, they are used in the macro definitions, which are generated from Language Definitions, and are not intended for external use:

Table 7. Antz tasks and data types for macro definitions

Class name	Tag	Description
Alloc	<antz:alloc>	An Ant data type that represents a data set allocation.
Concat	<antz:concat>	An Ant data type that represents a data set concatenation.

Table 7. Antz tasks and data types for macro definitions (continued)

Class name	Tag	Description
Executable	<antz:mvsexec>	An Ant task that invokes an MVS module specified in a Translator contained in a Language Definition.

Note: All custom Antz data types and tasks are defined in the antlib:com.ibm.teamz.build.ant namespace. This namespace must be declared in any build script that uses Antz tasks.

Antz Custom Datatypes

The BuildableResourceCollection, ComponentSelector, and ProjectSelector custom data types provided by Antz, with parameters and examples, are described below.

BuildableResourceCollection Custom Datatype

A BuildableResourceCollection is an Ant ResourceCollection that represents a set of buildable resources. Buildable resources are defined as those repository workspace assets that are associated with a Language Definition. A BuildableResourceCollection is built based on the content of a buildable resource XML file. This file is generally produced by the Build Forge Service preprocessor when a build is requested.

Note that a BuildableResourceCollection can serve as an Ant resource selection container, so resource selector tags can be nested within the collection's tag in order to select a subset of the buildable resources defined in the XML file. However, since the current implementation of buildable resources do not support file size or file date as their attributes, resource selector tags that use those attributes cannot be nested.

A BuildableResourceCollection is defined using the <antz_buildableset/> tag.

BuildableResourceCollection Parameter

Table 8. BuildableResourceCollection Parameter

Attribute	Description
buildableList	Path to the buildable resource XML file to be used when creating the BuildableResourceCollection

BuildableResourceCollection Examples

Create a resource collection based on the content of the file whose path is stored in the property ${buildablefile} and assigns it the reference id ″mySet″.

```
<antz:buildableset id="mySet" buildableList="${buildablefile}"/>
```

Creates a resource collection that includes only the files that:

- Exist on the MVS file system
- Were defined within the project ″User Management″
- That are members defined in the MVS dataset DEARTH.COBOL.SET1

```
<antz:buildableset id="sampleset" buildableList="${buildablefile}">
  <rsel:exists/>
  <antz:projectselector name="User Management"/>
  <rsel:name name="DEARTH.COBOL.SET1(*)"/>
</antz:buildableset>
```

ComponentSelector Custom Datatype

A component selector is a custom resource selector that allows the resources to be included in a BuildableResourceCollection based on the name of the component that defines the resource. The <antz:componentselector> tag is used to specify a component selector.

ComponentSelector Parameters

Table 9. ComponentSelector Parameters

Attribute	Description
name	The name of the project to which the resource's component name will be compared. Value can contain the special characters ″*″, representing a set of 0 or more characters, and ″?″, representing any character.
caseSensitive	Indicates whether the comparison should be performed in a case sensitive manner. Default is true.

ComponentSelector Example

Create a resource collection containing only those buildable resources whose component name begins with ″myComponent″. Case is not considered.

```
<antz:buildableset id="mySet" buildableList="${buildablefile}">
    <antz:componentselector name="myComponent*" caseSensitive="false"/>
</antz:buildableset>
```

ProjectSelector Custom Datatype

A project selector is a custom resource selector that allows the resources to be included in a BuildableResourceCollection based on the name of the project that defines the resource. The <antz:projectselector> tag is used to specify a project selector.

ProjectSelector Parameters

Table 10. ProjectSelector Parameters

Attribute	Description
name	The name of the project to which the resource's project name will be compared. Value can contain the special characters ″*″, representing a set of 0 or more characters, and ″?″, representing any character.
caseSensitive	Indicates whether the comparison should be performed in a case sensitive manner. Default is true.

ProjectSelector Example

Create a resource collection containing only those buildable resources whose project name begins with ″myProject″. Case is not considered.

```
<antz:buildableset id="mySet" buildableList="${buildablefile}"> ">
    <antz:projectselector name="myProject*" caseSensitive="false"/>
</antz:buildableset>
```

Antz Compile Custom Data Task

The Compile custom data task provided by Antz, with examples, is described below.

Compile Custom Data Task

A Compile task is a custom Ant task that compiles a BuildableResourceCollection that is provided as its child element. Internally, it invokes an Ant macro that corresponds to the Language Definition associated with a buildable resource.

Compile Example

This example code iterates over all buildable resources whose name matches "*LINK(*)", and executes the Antz macro that was generated for the Language Definition associated with the resource.

```
<antz:compile>
    <antz:buildableset buildableList="${buildfile}">
        <rsel:name name="*LINK(*)"/>
    </antz:buildableset>
</antz:compile>
```

Appendix F. Job Monitor security

This appendix contains information to assist you in setting up security for the Job Monitor.

Security considerations

The security mechanisms used by Job Monitor relies on the file system it resides in being secure. This implies that only trusted system administrators should be able to update the program libraries and configuration files.

JES Security

Job Monitor allows clients access to the JES spool. The server provides basic access limitations, which can be extended with the standard spool file protection features of your security product. Operator actions (Hold, Release, Cancel, and Purge) against spool files are done through an EMCS console, for which conditional permits must be set up.

Actions against jobs - target limitations

Job Monitor does not provide users full operator access to the JES spool. Only the Hold, Release, Cancel, and Purge commands are available, and by default, only for spool files owned by the user. The commands are issued by selecting the appropriate option in the client menu structure (there is no command prompt). The scope of the commands can be widened, using security profiles to define for which jobs the commands are available.

Similar to the SDSF **SJ** action character, Job Monitor also supports the Show JCL command to retrieve the JCL that created the selected job output, and show it in an editor window. Job Monitor retrieves the JCL from JES, making it a useful function for situations in which the original JCL member is not easily located.

Table 11. Job Monitor console commands

Action	JES2	JES3
Hold	`$Hx(jobid)` with x = {J, S or T}	`*F,J=jobid,H`
Release	`$Ax(jobid)` with x = {J, S or T}	`*F,J=jobid,R`
Cancel	`$Cx(jobid)` with x = {J, S or T}	`*F,J=jobid,C`
Purge	`$Cx(jobid),P` with x = {J, S or T}	`*F,J=jobid,C`
Show JCL	`not applicable`	`not applicable`

The available JES commands listed in Table 11 are by default limited to jobs owned by the user. This can be changed with the `LIMIT_COMMANDS` directive, as documented in “BLZJCNFG, Job Monitor configuration file” on page 49.

Table 12. Job Monitor command permission matrix

LIMIT_COMMANDS	**User**	**Other**
USERID (default)	Allowed	Not allowed
LIMITED	Allowed	Allowed only if explicitly permitted by security profiles
NOLIMIT	Allowed	Allowed if permitted by security profiles or when the JESSPOOL class is not active

JES uses the JESSPOOL class to protect SYSIN/SYSOUT data sets. Similar to SDSF, Job Monitor extends the use of the JESSPOOL class to protect job resources as well.

If LIMIT_COMMANDS is not USERID, then Job Monitor will query for permission to the related profile in the JESSPOOL class, as shown in the following table.

Table 13. Extended JESSPOOL profiles

Header	JESSPOOL profile	Required access
Hold	nodeid.userid.jobname.jobid	ALTER
Release	nodeid.userid.jobname.jobid	ALTER
Cancel	nodeid.userid.jobname.jobid	ALTER
Purge	nodeid.userid.jobname.jobid	ALTER
Show JCL	nodeid.userid.jobname.jobid.JCL	READ

Use the following substitutions in the preceding table:

nodeid	NJE node ID of the target JES subsystem
userid	Local user ID of the job owner
jobname	Name of the job
jobid	JES job ID

If the JESSPOOL class is not active, then there is different behavior for the LIMITED and NOLIMIT value of LIMIT_COMMANDS, as described in "BLZJCNFG, Job Monitor configuration file" on page 49. The behavior is identical when JESSPOOL is active, since the class, by default, denies permission if a profile is not defined.

Actions against jobs - execution limitations

The second phase of JES spool command security, after specifying the permitted targets, includes the permits needed to actually execute the operator command. This execution authorization is enforced by the z/OS and JES security checks.

Note that Show JCL is not an operator command like the other Job Monitor commands (Hold, Release, Cancel and Purge), so the limitations below do not apply because there is no further security check.

Job Monitor issues all JES operator commands requested by a user through an extended MCS (EMCS) console, whose name is controlled with the CONSOLE_NAME directive, as documented in "BLZJCNFG, Job Monitor configuration file" on page 49.

This setup allows the security administrator to define granular command execution permits using the OPERCMDS and CONSOLE classes.

- In order to use an EMCS console, a user must have (at least) READ authority to the MVS.MCSOPER.console-name profile in the OPERCMDS class. Note that if no profile is defined, the system will grant the authority request.
- In order to execute a JES operator command, a user must have sufficient authority to the JES%.** (or more specific) profile in the OPERCMDS class. Note that if no profile is defined, or the OPERCMDS class is not active, JES will fail the command.
- The security administrator can also require that a user must use Job Monitor when executing the operator command by specifying WHEN(CONSOLE(JMON)) on the **PERMIT** definition. The CONSOLE class must be active for this setup to work. Note that the CONSOLE class being active is sufficient; no profiles are checked for EMCS consoles.

Assuming the identity of the Job Monitor server by creating a JMON console from a TSO session is prevented by your security software. Even though the console can be created, the point of entry is different (Job Monitor versus TSO). JES commands issued from this console will fail the security check, if your security is set up as documented in this publication and the user does not have authority to JES commands through other means.

Note that Job Monitor cannot create the console when a command must be executed if the console name is already in use. To prevent this, the system programmer can set the GEN_CONSOLE_NAME=ON directive in the Job Monitor configuration file or the security administrator can define security profiles to stop TSO users from creating a console. The following sample RACF commands prevent everyone (except those permitted) from creating a TSO or SDSF console:

- RDEFINE TSOAUTH CONSOLE UACC(NONE)
- PERMIT CONSOLE CLASS(TSOAUTH) ACCESS(READ) ID(#userid)
- RDEFINE SDSF ISFCMD.ODSP.ULOG.* UACC(NONE)
- PERMIT ISFCMD.ODSP.ULOG.* CLASS(SDSF) ACCESS(READ) ID(#userid)

Note: Without being authorized for these operator commands, users will still be able to submit jobs and read job output through the Job Monitor, if they have sufficient authority to possible profiles that protect these resources (like those in the JESINPUT, JESJOBS and JESSPOOL classes).

Refer to *Security Server RACF Security Administrator's Guide (SA22-7683)* for more information on operator command protection.

Access to spool files

Job Monitor allows browse access to all spool files by default. This can be changed with the LIMIT_VIEW directive, as documented in "BLZJCNFG, Job Monitor configuration file" on page 49.

Table 14. Job Monitor browse permission matrix

	Job owner	
LIMIT_VIEW	**User**	**Other**
USERID	Allowed	Not allowed

Table 14. Job Monitor browse permission matrix (continued)

	Job owner	
LIMIT_VIEW	**User**	**Other**
NOLIMIT (default)	Allowed	Allowed if permitted by security profiles or when the JESSPOOL class is not active

To limit users to their own jobs on the JES spool, define the "LIMIT_VIEW=USERID" statement in the Job Monitor configuration file, BLZJCNFG. If they need access to a wider range of jobs, but not all, use the standard spool file protection features of your security product, like the JESSPOOL class.

When defining further protection, keep in mind that Job Monitor uses SAPI (SYSOUT application program interface) to access the spool. This implies that the user needs at least UPDATE access to the spool files, even for browse functionality. This requisite does not apply if you run z/OS 1.7 (z/OS 1.8 for JES3) or higher. Here READ permission is sufficient for browse functionality.

Refer to *Security Server RACF Security Administrator's Guide (SA22-7683)* for more information on JES spool file protection.

Job Monitor configuration file

The directives in the BLZJCNFG Job Monitor configuration file impact the security setup. The security administrator and systems programmer can decide what the settings should be for the following directives.

- LIMIT_COMMANDS={USERID | LIMITED | NOLIMIT}

  Define against which jobs actions can be done (excluding browse and submit). For more information, see "Actions against jobs - target limitations" on page 95.
- LIMIT_VIEW={USERID | NOLIMIT}

  Define which spool files can be browsed. For more information, see "Access to spool files" on page 97.

Note: Details on the BLZJCNFG directives are available in "BLZJCNFG, Job Monitor configuration file" on page 49.

Security definitions

The following sections describe the required steps, optional configuration and possible alternatives.

- "Define the Job Monitor started task."
- "Define JES command security" on page 99.

Refer to the *RACF Command Language Reference (SA22-7687)*, for more information on RACF commands.

Define the Job Monitor started task

The following sample RACF commands create the BLZJMON, started tasks, with protected user IDs (STCJMON) and group STCGROUP assigned to them. Replace the #group-id and #user-id-* placeholders with valid OMVS IDs.

- ```
  ADDGROUP STCGROUP OMVS(GID(#group-id))
  DATA('GROUP WITH OMVS SEGMENT FOR STARTED TASKS')
  ```
  ```

- `ADDUSER STCJMON DFLTGROUP(STCGROUP) NOPASSWORD NAME('JOB MONITOR') OMVS(UID(#user-id-jmon) HOME(/tmp) PROGRAM(/bin/sh) NOASSIZEMAX) DATA('RATIONAL TEAM CONCERT')`
- `RDEFINE STARTED BLZJJCL.* DATA('JOB MONITOR') STDATA(USER(STCJMON) GROUP(STCGROUP) TRUSTED(NO))`
- `SETROPTS RACLIST(STARTED) REFRESH`

Note: Ensure that the started task user ID is protected by specifying the `NOPASSWORD` keyword.

Define JES command security

Job Monitor issues all JES operator commands requested by a user through an extended MCS (EMCS) console, whose name is controlled with the `CONSOLE_NAME` directive, as documented in "BLZJCNFG, Job Monitor configuration file" on page 49.

The following sample RACF commands give Job Monitor users conditional access to a limited set of JES commands (Hold, Release, Cancel, and Purge). Users only have execution permission if they issue the commands through Job Monitor. Replace the #console placeholder with the actual console name.

- `RDEFINE OPERCMDS MVS.MCSOPER.#console UACC(READ) DATA('RATIONAL TEAM CONCERT'))`
- `RDEFINE OPERCMDS JES%.** UACC(NONE)`
- `PERMIT JES%.** CLASS(OPERCMDS) ACCESS(UPDATE) WHEN(CONSOLE(JMON)) ID(*)`
- `SETROPTS RACLIST(OPERCMDS) REFRESH`

Note:

1. Usage of the console is permitted if no `MVS.MCSOPER.#console` profile is defined.
2. The `CONSOLE` class must be active for `WHEN(CONSOLE(JMON))` to work, but there is no actual profile check in the `CONSOLE` class for EMCS consoles.
3. Do not replace `JMON` with the actual console name in the `WHEN(CONSOLE(JMON))` clause. The `JMON` keyword represents the point-of-entry application, not the console name.

Attention: Defining JES commands with universal access `NONE` in your security software might impact other applications and operations. Test this before activating it on a production system.

Table 15 and Table 16 on page 100show the operator commands issued for JES2 and JES3, and the discrete security profiles that can be used to protect them.

Table 15. JES2 Job Monitor operator commands

Action	Command	OPERCMDS profile	Required access

Hold	`$Hx(jobid)` with x = {J, S or T}	`jesname.MODIFYHOLD.BAT`
		`jesname.MODIFYHOLD.STC`
		`jesname.MODIFYHOLD.TSU`
			`UPDATE`

Release	`$Ax(jobid)` with x = {J, S or T}	`jesname.MODIFYRELEASE.BAT`
		`jesname.MODIFYRELEASE.STC`
		`jesname.MODIFYRELEASE.TSU`
			`UPDATE`

Cancel	`$Cx(jobid)` with x = {J, S or T}	`jesname.CANCEL.BAT`
		`jesname.CANCEL.STC`
		`jesname.CANCEL.TSU`
			`UPDATE`

Table 15. JES2 Job Monitor operator commands (continued)

Action	Command	OPERCMDS profile	Required access
Purge	$Cx(jobid),P with x = {J, S or T}	jesname.CANCEL.BAT jesname.CANCEL.STC jesname.CANCEL.TSU	UPDATE

Table 16. JES3 Job Monitor operator commands

Action	Command	OPERCMDS profile	Required access
Hold	*F,J=jobid,H	jesname.MODIFY.JOB	UPDATE
Release	*F,J=jobid,R	jesname.MODIFY.JOB	UPDATE
Cancel	*F,J=jobid,C	jesname.MODIFY.JOB	UPDATE
Purge	*F,J=jobid,C	jesname.MODIFY.JOB	UPDATE

Note:

1. The Hold, Release, Cancel, and Purge JES operator commands, and the Show JCL command, can only be executed against spool files owned by the client user ID, unless LIMIT_COMMANDS= with value LIMITED or NOLIMIT is specified in the Job Monitor configuration file. Refer to "Actions against jobs - target limitations" on page 95 for more information on this.
2. Users can browse any spool file, unless LIMIT_VIEW=USERID is defined in the Job Monitor configuration file. Refer to "Access to spool files" on page 97 for more information on this.
3. Without being authorized for these operator commands, users will still be able to submit jobs and read job output through Job Monitor, if they have sufficient authority to possible profiles that protect these resources (like those in the JESINPUT, JESJOBS and JESSPOOL classes).

Assuming the identity of the Job Monitor server by creating a JMON console from a TSO session is prevented by your security software. Even though the console can be created, the point of entry is different (Job Monitor versus TSO). JES commands issued from this console will fail the security check, if your security is set up as documented in this publication and the user does not have authority to the JES commands through other means.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
20 Maguire Road
Lexington, MA 02421-3112
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

These terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

Adobe, Acrobat, Portable Document Format (PDF), PostScript, and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Program Number: 5724-V82

Printed in USA