z/OS

z/OS Batch Runtime: Planning and User's Guide

SA23-2270-01

Note

Before using this information and the product it supports, read the information in “Notices” on page 51.

This edition applies to version 1, release 13, modification 0 of IBM z/OS (product number 5694-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SA23-2270-00.

Contents

Figures

Tables

About this information

This publication describes the IBM® z/OS Batch Runtime component of z/OS. z/OS Batch Runtime provides the ability to update the DB2® database from both COBOL and Java in a single transaction.

This publication is organized as follows:

- Chapter 1, “Overview and planning of z/OS Batch Runtime,” on page 1. This chapter describes overview information for z/OS Batch Runtime and how to invoke the program.
- Chapter 2, “Invoking z/OS Batch Runtime,” on page 5. This chapter describes how to invoke the z/OS Batch Runtime program through the job control language (JCL).
- Chapter 3, “Defining connectivity for the database,” on page 17. This chapter describes planning connectivity for z/OS Batch Runtime.
- Chapter 4, “Application interfaces for z/OS Batch Runtime,” on page 19. This chapter describes application programming interfaces for z/OS Batch Runtime including: options, support elements for the Java Database Connectivity (JDBC) and DB2 programs, environment variables, completion codes, and any applicable API.
- Chapter 6, “Troubleshooting for z/OS Batch Runtime,” on page 47. This chapter describes diagnostics and troubleshooting procedures for z/OS Batch Runtime.

Who should use this information

This publication is intended for experienced Java and COBOL programmers who are familiar with DB2 and plan, develop, and test applications that run on z/OS. It describes how to improve interoperability between COBOL and Java applications by allowing you to share a local DB2 attachment in a single hybrid Java COBOL application. Advanced knowledge of the Java Native Interface (JNI), COBOL programming, and DB2 is required.

Note: All examples in this publication are for illustration purposes only. You must replace any example or code parameters with the correct specifications for your installation.

Where to find more information

Where necessary, this publication references information in other publications, using shortened versions of the publication title. For complete titles and order numbers of the publications for all products that are part of z/OS, see *z/OS Information Roadmap*.

Information updates on the web

For the latest information updates that have been provided in PTF cover letters and information APARs for z/OS®, see the online information at:

`http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/ZDOCAPAR`

This information is updated weekly and lists changes before they are incorporated into z/OS publications.

The z/OS Basic Skills Information Center

The z/OS Basic Skills Information Center is a Web-based information resource intended to help users learn the basic concepts of z/OS, the operating system that runs most of the IBM mainframe computers in use today. The Information Center is designed to introduce a new generation of Information Technology professionals to basic concepts and help them prepare for a career as a z/OS professional, such as a z/OS system programmer.

Specifically, the z/OS Basic Skills Information Center is intended to achieve the following objectives:

- Provide basic education and information about z/OS without charge
- Shorten the time it takes for people to become productive on the mainframe
- Make it easier for new people to learn z/OS.

To access the z/OS Basic Skills Information Center, open your Web browser to the following Web site, which is available to all users (no login required): http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp

How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or give us any other feedback that you might have.

Use one of the following methods to send us your comments:

1. Send an email to mhvrcfs@us.ibm.com
2. Visit the Contact z/OS web page at http://www.ibm.com/systems/z/os/zos/webqs.html
3. Mail the comments to the following address:

   IBM Corporation
   Attention: MHVRCFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   U.S.A.
4. Fax the comments to us as follows:

   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address
- Your email address
- Your telephone or fax number
- The publication title and order number:

  z/OS V1R13.0 Batch Runtime Planning and User's Guide
  SA23-2270-00
- The topic and page number related to your comment
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you submit.

If you have a technical problem

Do not use the feedback methods listed above. Instead, do one of the following:

- Contact your IBM service representative
- Call IBM technical support
- Visit the IBM support portal at http://www.ibm.com/systems/z/support/

Summary of Changes

This document contains terminology, maintenance, and editorial changes to improve consistency and retrievability. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Changes made in z/OS Version 1 Release 13 (as of October 2011)

The book contains information that was previously presented in *z/OS Batch Runtime Planning and User's Guide*, SA23-2270-00, which supports z/OS Version 1 Release 13.

Changed information:

- The level of Java required for z/OS Batch Runtime has been updated to Java 6.0.1 throughout this document, including updates to examples.
- Additional graphics and examples have been added throughout the document.

Chapter 1. Overview and planning of z/OS Batch Runtime

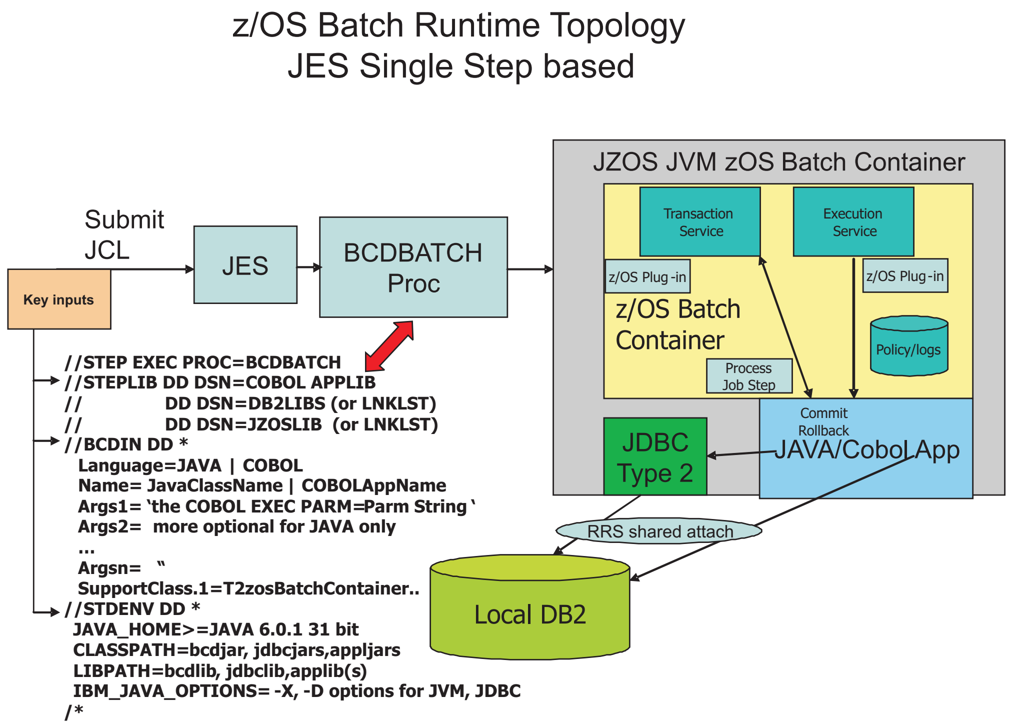
In today's z/OS environment, many installations want to re-engineer their existing native z/OS COBOL applications to incorporate the Java language. By doing so, they can keep their heritage of existing z/OS COBOL batch applications, while taking advantage of the larger developer skill base and many language features of Java. As such, there is a requisite need to share a local DB2 for z/OS attachment across the Java and COBOL language boundary. This enables mixed language programs to process DB2 for z/OS requests in the same unit of work (UOW). When these batch application suites are re-engineered or updated, they should also allow transparent local DB2 for z/OS access from both COBOL and Java to the following programs:

- Embedded Structured Query Language (SQL) DB2 access, which is used in Enterprise COBOL
- Java Database Connectivity (JDBC) for Dynamic SQL
- Embedded Structured Query Language for Java (SQLJ)

z/OS Batch Runtime allows for this interoperability between COBOL applications and Java applications that run on z/OS. It is a program designed to provide a managed environment that enables shared access to a DB2 connection by both COBOL and Java programs. Updates to DB2 are committed in a single transaction. (Note that updates to multiple databases are not supported.)

Figure 1 on page 2 shows a high-level overview of the z/OS Batch Runtime environment. The batch container performs the initialization that sets up the environment for COBOL, Java, and DB2 interoperability. This includes the following tasks:

- Setting up the proper Language Environment® for the COBOL programs to run
- Setting up the job step under the umbrella of a Resource Recovery Services (RRS)-managed global transaction
- Initiating the DB2 JDBC driver in this special “BatchContainer” mode
- Invoking the DB2 JDBC driver to create a DB2 connection and attachment thread
- Invoking the primary COBOL or Java application after the environment is properly initialized.



```
//STEP EXEC PROC=BCDBATCH
//STEPLIB DD DSN=COBOL APPLIB
//             DD DSN=DB2LIBS (or LNKLST)
//             DD DSN=JZOSLIB  (or LNKLST)
//BCDIN DD *
  Language=JAVA | COBOL
  Name= JavaClassName | COBOLAppName
  Args1= ‘the COBOL EXEC PARM=Parm String ‘
  Args2=  more optional for JAVA only
  ...
  Argsn=   “
  SupportClass.1=T2zosBatchContainer..
//STDENV DD *
 JAVA_HOME>=JAVA 6.0.1 31 bit
 CLASSPATH=bcdjar, jdbcjars,appljars
 LIBPATH=bcdlib, jdbclib,applib(s)
 IBM_JAVA_OPTIONS= -X, -D options for JVM, JDBC
/*
```

Figure 1. Overview of the z/OS Batch Runtime environment

Requirements for z/OS Batch Runtime

z/OS Batch Runtime requires the following programs:

- IBM 31-bit SDK for z/OS, Java Technology Edition, V6.0.1 (5655-R31) (For details, see “Configuring Java” on page 5.
- IBM Enterprise COBOL Version 4.2
- One of the following:
  - – DB2 V9 with PTF UK62190 for JDBC 3.0 specification level, or PTF UK62191 for JDBC 4.0 specification level
  - – DB2 V10 with PTF UK62141 for JDBC 3.0 specification level, or PTF UK62145 for JDBC 4.0 specification level

For more information about these required programs, see the appropriate reference listed in Table 1.

Table 1. Summary of reference information for required programs

For information about	Refer to
Java	`http://www.ibm.com/systems/z/os/zos/tools/java/`
IBM Enterprise COBOL Version 4 Release 2	`http://www.ibm.com/software/awdtools/cobol/zos/library/`
DB2	`http://www.ibm.com/software/data/db2/zos/family/`

Planning for z/OS Batch Runtime

When planning use of z/OS Batch Runtime, a good application to consider using is a native procedural z/OS COBOL application that you want to functionally enhance with Java method calls. The entire application code must be single threaded. Also, see Chapter 5, “Application structure and build considerations,” on page 25 for more information.

Chapter 2. Invoking z/OS Batch Runtime

The z/OS Batch Runtime is established by launching the Java program `com.ibm.zos.batch.container.BCDBatchContainer` with the proper configuration and environment settings that allows your Java and COBOL programs to be invoked with the correct arguments. The JZOS launcher, a component of the IBM JDK for z/OS, is used to establish the environment and pass control to z/OS Batch Runtime which, in turn, will launch your COBOL or Java program and provide necessary services. To facilitate the use of z/OS Batch Runtime, z/OS includes:

- Environment tailoring shell scripts: `bcdconfig.sh` and `bcdconfigend.sh in /usr/lpp/bcp`
- A JCL procedure to be invoked by batch jobs: BCDPROC in SYS1.PROCLIB
- A sample batch job to use BCDPROC: BCDBATCH in SYS1.SAMPLIB

Configuring Java

You must configure the CLASSPATH and LIBPATH variables with the list of Java archive (JAR) files and dynamic link library (DLL) files that are required to run both the z/OS Batch Runtime and the application. z/OS Batch Runtime is itself a Java application and uses the JZOS toolkit to launch the JVM. You should tailor the z/OS Batch Runtime sample BCDBATCH JCL and the environment variables it provides.

Additionally, JZOS defines several environment variables that allow you to control the Java options that JZOS uses when it creates the JVM and main method program arguments. Find these options and complete information in *JZOS Batch Launcher and Toolkit function in IBM SDK for z/OS*, SA23-2245, at `www.ibm.com/systems/z/os/zos/tools/java/products/jzos/overview.html`.

Note: Although JZOS also defines environment variables that allow you to control the encoding of output, z/OS Batch Runtime only supports EBCDIC file encoding.

Improving Java start up time

For short-running jobs, improving Java start up time is important. This is especially true when running numerous small Java batch jobs, as the Java start up elapsed time and CPU time may affect performance. Using the following Java options can make it possible to reduce the Java startup times for applications that frequently start a new JVM :

- -Xquickstart Java option

  Note: Quickstart may improve startup time for short running jobs, but it may degrade performance of long running applications.
- Shared classes and AOT Java options

For more details about this topic as well as the latest considerations for using Java, performance information, hints and tips, and information about developing and running applications see:

`http://www.ibm.com/systems/z/os/zos/tools/java`

Java environment variables for z/OS Batch Runtime

Java applications use the following environment variables for z/OS Batch Runtime that are specified in the JCL:

- JAVA_HOME

- CLASSPATH
- LIBPATH
- IBM_JAVA_OPTIONS

See “Procedure for modifying the BCDBATCH job” on page 9 for examples of how to specify these environment variables.

JAVA_HOME

The application must set the JAVA_HOME environment variable to a minimum level of JAVA 6.0.1 and specify 31-bit only.

CLASSPATH

The application must set the CLASSPATH to include the .JAR files for z/OS Batch Runtime, the DB2 driver for JDBC (DB2 JCC), and the application. To do so, use the CLASSPATH environment variable specified in the BCDBATCH JCL procedure.

The configuration script automatically updates the CLASSPATH for z/OS Batch Runtime .jar files, based on the exported BCD_HOME variable in the BCDBATCH JCL procedure.

LIBPATH

In the BCDBATCH JCL procedure, the application must set LIBPATH to the location of the DLLs for z/OS Batch Runtime, DB2 JCC, and any that are associated with application. The configuration script performs the function.

IBM_JAVA_OPTIONS

This environment variable is a concatenation of the IBM JVM runtime options, which are typically prefixed with -X, and any Java system properties, which are prefixed with -D. This can include, for example, the JVM heap size runtime option and the DB2 package list system property.

31-bit support

z/OS Batch Runtime supports only 31-bit applications; you must use the 31-bit JVM.

Main JCL statements needed for BCDBATCH

This section of the documentation uses reference keys, such as **1**, **2**, to match the instructions with the sample JCL.

z/OS Batch Runtime supplies a sample BCDBATCH job which you modify to suit your application. Table 2 summarizes the main JCL statements for the BCDBATCH job. “Procedure for modifying the BCDBATCH job” on page 9 contains complete steps to modifying the sample BCDBATCH job.

Table 2. JCL summary for BCDBATCH job

JCL statement	Explanation

1
`//BCDBATCH  JOB  (1),'name'`
`//BATCH EXEC BCDPROC,REGION=0M,LOGLVL='+I'`
	The JCL that invokes z/OS Batch Runtime. Throughout this publication, the JCL used to invoke the z/OS Batch Runtime is referred to as the BCDBATCH job. Use any job name that is acceptable to your installation.

Table 2. JCL summary for BCDBATCH job (continued)

JCL statement	Explanation

2
`//*STEPLIB DD  DSN=hlq.yourapp.loadlib,DISP=SHR`
`//*        DD  DSN=hlq.jzos.loadlib,DISP=SHR`
	Add any load libraries your application requires to the STEPLIB; for example, this could be the data set containing your COBOL application load modules. If the JZOS Java launcher is not installed in the LNKLST, add a STEPLIB for it. For more information about installing JZOS, see the JZOS Java Launcher and Toolkit Overview at `www.ibm.com/systems/z/os/zos/tools/java/`.
	Any COBOL application modules must be in either the //STEPLIB concatenation or added to a STEPLIB environment variable in //STDENV DD *. Do not use LIBPATH for starting a COBOL application.

3 `//STDENV DD *`	Specifies the environment variables used for this run, including JAVA_HOME, CLASSPATH, and LIBPATH.
`//BCDIN DD *`	Specifies a file containing the batch configuration options. Note that some support elements obtain their options from Java system properties. See “JCL for BCDIN configurations options” on page 11 for more information.

JCL for the BCDBATCH job

A current sample of BCDBATCH job for z/OS Batch Runtime is in SYS1.SAMPLIB. For convenience and planning purposes, this documentation contains the following “Sample BCDBATCH JCL,” “Procedure for modifying the BCDBATCH job” on page 9, and “Sample BCDPROC to invoke z/OS Batch Runtime” on page 14.

Note: All examples in this publication are for illustration purposes only. You must replace any example or code parameters with the correct specifications for your installation.

Sample BCDBATCH JCL

Figure 2 on page 8 is an example of JCL procedure for running the sample BCDBATCH job.

```
1
//BCDBATCH  JOB  (1),'name'
//BATCH  EXEC  BCDPROC,REGION=0M,LOGLVL='+I'
//*
//*********************************************************************
//* Update: Add the load libraries your application requires,         *
//*         such as the data set containing your COBOL                *
//*         application load modules to the STEPLIB.                  *
//*                                                                   *
//*         If the JZOS Java launcher has not been installed in       *
//*         the lnklst, add a steplib for it.                         *
//*********************************************************************
2
//*STEPLIB DD  DSN=hlq.yourapp.loadlib,DISP=SHR
//*        DD  DSN=hlq.jzos.loadlib,DISP=SHR
//*
//*
3
//STDENV DD *
#
#--------------------------------------------------------------------
#  UPDATE: Installation path for Batch Runtime.
#          Note: because the Batch Runtime is a component of z/OS,
#          the installation defaults to /usr/lpp/bcp
#--------------------------------------------------------------------
export BCD_HOME=/usr/lpp/bcp
# 4
#--------------------------------------------------------------------
#  UPDATE: Installation path for Java.
#--------------------------------------------------------------------
export JAVA_HOME=/usr/lpp/java/J6.0.1
#
# 5
#--------------------------------------------------------------------
#  The following runs the z/OS Batch Runtime configuration script.
#  This script processes the exported environment variables that
#  were defined above.
#--------------------------------------------------------------------
. $BCD_HOME/bcdconfig.sh
#
# 6
#--------------------------------------------------------------------
#  UPDATE: JDBC home directory, jar files, and DLLs.
#--------------------------------------------------------------------
#JDBC_HOME=/usr/lpp/db2910_jdbc
#CLASSPATH="$CLASSPATH":$JDBC_HOME/classes/db2jcc.jar
#CLASSPATH="$CLASSPATH":$JDBC_HOME/classes/db2jcc_javax.jar
#export CLASSPATH="$CLASSPATH"
#
#LIBPATH="$LIBPATH":$JDBC_HOME/lib
#export LIBPATH="$LIBPATH"
```

Figure 2. Example: BCDBATCH JCL procedure (Part 1 of 2)

```
#
# 7
#----------------------------------------------------------------------
#   UPDATE: Add your application jar files to the CLASSPATH here.
#----------------------------------------------------------------------
#CLASSPATH="$CLASSPATH":/your/extra/app.jar
#CLASSPATH="$CLASSPATH":/your/extra/app2.jar
#export CLASSPATH="$CLASSPATH"
#
# 8
#----------------------------------------------------------------------
#   UPDATE: Add your application libraries to the LIBPATH here.
#   The LIBPATH defines points to any application-defined DLLs,
#   which may include Java Native Interface (JNI) routines.
#----------------------------------------------------------------------
#LIBPATH="$LIBPATH":/your/extra/lib
#LIBPATH="$LIBPATH":/your/extra/lib2
#export LIBPATH="$LIBPATH"
#
# 9
#----------------------------------------------------------------------
#   UPDATE: Uncomment to enable z/OS Batch Runtime tracing for diagnosis.
#----------------------------------------------------------------------
#IJO="$IJO -Dcom.ibm.zos.batch.container.BCDTraceConfig.trace=all"
#
# 10
#----------------------------------------------------------------------
#   UPDATE: Uncomment and add any additional JVM options here.
#----------------------------------------------------------------------
#IJO="-Xms256m -Xmx512m"
#
# 11
#----------------------------------------------------------------------
#   UPDATE: Uncomment and add JDBC options here.
#----------------------------------------------------------------------
#IJ0=$IJ0 -Ddb2.jcc.ssid=XXXX -Ddb2.jcc.pkList=NULLID.*,COBOLPKG.*"
#
# 12
#----------------------------------------------------------------------
#   Exports JVM options set above.
#----------------------------------------------------------------------
export IBM_JAVA_OPTIONS="$IJO "
#
# 13
#----------------------------------------------------------------------
#  The following runs the z/OS Batch Runtime configuration completion
#  script.  This command must be last in the STDENV file.
#----------------------------------------------------------------------
. $BCD_HOME/bcdconfigend.sh
#
//
```

Figure 2. Example: BCDBATCH JCL procedure (Part 2 of 2)

Procedure for modifying the BCDBATCH job

The following JCL procedure summarizes the key statements to modify in the BCDBATCH job (see Figure 2 on page 8) that invokes z/OS Batch Runtime.

__ 1 Modify the JOB and EXEC statements to add any parameters required by your installation.

For example in the following statement, `BCDPROC` is the batch container JCL procedure.

```
//BATCH EXEC BCDPROC,REGION=0M
```

For details and options, including the symbolic to override defaults, see “Sample BCDPROC to invoke z/OS Batch Runtime” on page 14.

__ **2** For the STEPLIB statement, specify any load libraries that the application requires (for example, the data set that contains your COBOL application load modules) for DSN=, where *hlq.yourapp.loadlib* is the name:

```
//*STEPLIB DD DSN=hlq.yourapp.loadlib,DISP=SHR
//  DD DSN=hlq.jzos.loadlib,DISP=SHR
```

This may include requisite DB2 and COBOL libraries that are not in LNKLST but are loaded during program execution. Note that any COBOL modules that are bound as DLLs should usually be found through the LIBPATH environment variable.

The batch container uses the Java SDK JZOS launcher utility. If you installed the Java SDK using SMP/E, JZOS is installed in the LNKLST. However, if you did not use SMP/E, you must install the JZOS launcher into a data set, and add that to your STEPLIB concatenation.

For more information about installing JZOS, see the JZOS Java Launcher and Toolkit Overview at `www.ibm.com/systems/z/os/zos/tools/java/`

__ **3** Update the installation paths for z/OS Batch Runtime. To tailor the runtime environment, use the `//STDENV DD` statement in the BCDBATCH JCL to define a shell script. The batch container processes the exported `BCD_HOME` environment variable referenced by the script as the installation path for z/OS Batch Runtime (default is `/usr/lpp/bcp`).

__ **4** Update the installation path for Java to the correct level of Java:

```
export JAVA_HOME=/usr/lpp/java/J6.0.1
```

__ **5** Run the z/OS Batch Runtime configuration shell script, bcdconfig, to process the exported environment variables you just defined:

```
. $BCD_HOME/bcdconfig.sh
```

To set up the batch container, you must use the . (dot) command to invoke the bcdconfig.sh.

__ **6** Update the JDBC home directory, jar files, and DLLs:

```
JDBC_HOME=/usr/lpp/db2910_jdbc
```

__ **7** Add additional application jar files to the CLASSPATH:

```
CLASSPATH="$CLASSPATH":/your/extra/app.jar
#CLASSPATH="$CLASSPATH":/your/extra/app2.jar
#export CLASSPATH="$CLASSPATH"
```

__ **8** Add your application DLLs to the LIBPATH directories:

```
LIBPATH="$LIBPATH":/your/extra/lib
LIBPATH="$LIBPATH":/your/extra/lib2
export LIBPATH="$LIBPATH"
```

__ **9** Enable tracing for z/OS Batch Runtime:

```
IJO="$IJO -Dcom.ibm.zos.batch.container.BCDTraceConfig.trace=all
```

__ **10** Add any additional JVM options:

```
IJO="-Xms256m -Xmx512m"
```

You may add, for example, the -Xquickstart option or any other -D or -X JVM runtime option you want to use.

__ **11** Add any additional JDBC options for the DB2 subsystem:

```
IJO="$IJO -Ddb2.jcc.ssid=XXXX -Ddb2.jcc.pkList=NULLID.*,COBOLPKG.*"
```

For more information about the Java Database Connectivity (JDBC) options, see *DB2 Application Programming Guide and Reference for Java* or the following Web site: `http://publib.boulder.ibm.com/epubs/pdf/dsnjvm01.pdf`

Do not specify -Dfile.encoding in the IBM_JAVA_OPTIONS string. z/OS Batch Runtime only supports the default z/OS file.encoding of IBM-1047.

__ **12** Export the JVM options:

```
export IBM_JAVA_OPTIONS="$IJO "
```

The IBM_JAVA_OPTIONS string must be set and exported before invoking the bcdconifgend.sh script.

__ **13** Run the following z/OS Batch Runtime completion script:

```
. $BCD_HOME/bcdconfigend.sh
```

Note: This script must always be run last in `STDENV`.

JCL for BCDIN configurations options

Use the `//BCDIN` JCL statement to control the z/OS Batch Runtime configuration options. Some support elements obtain their options from Java system properties.

When creating a configuration options file, the following rules apply:

- Options must appear in the `keyword=value` format
- Options must be coded in columns 1 through 71 of the record. Long options can be continued by coding a non-blank character in column 72 and continuing on the next line.
- Comment lines contain a # in column one.
- Blank lines are ignored.
- Options are case sensitive.
- When you specify an option more than once, the last occurrence is used.

Sample BCDIN File

Figure 3 on page 12 shows an example file that contains additional details and explanations. You can modify the sample as needed for individual jobs at your installation.

```
//*
//*********************************************************************
//*                                                                   *
//*            Batch Runtime Options                                  *
//*                                                                   *
//* Syntax rules for specifying options:                              *
//*                                                                   *
//*   1. Options are specified in keyword=value format.               *
//*                                                                   *
//*   2. Options are coded using columns 1-71.                        *
//*                                                                   *
//*      Long options may be continued by coding a non-blank          *
//*      character in column 72 and continuing on the next line.      *
//*                                                                   *
//*   3. Comment lines contain a # in column 1.                       *
//*                                                                   *
//*   4. Blank lines are ignored.                                     *
//*                                                                   *
//*   5. Option names are case sensitive.                             *
//*                                                                   *
//*   6. When the same option is specified more than once,            *
//*      the last occurrence of the option is used.                   *
//*                                                                   *
//*********************************************************************
//*
//BCDIN DD *
# 1
#---------------------------------------------------------------------*
# UPDATE: Uncomment the option corresponding to the language of
#         the application being launched
#---------------------------------------------------------------------*
#bcd.applicationLanguage=COBOL
#bcd.applicationLanguage=JAVA
#
# 2
#---------------------------------------------------------------------*
# UPDATE: The program name or fully qualified Java class name
#         of the application to be launched
#---------------------------------------------------------------------*
bcd.applicationName=your.application.name
```

Figure 3. Example: JCL BCDIN configuration options (Part 1 of 2)

```
#
# 3
#---------------------------------------------------------------------*
# UPDATE: Arguments to be passed to the launched application.
#
#         For Java applications, any number of arguments can be used.
#         Each argument is passed as an element of the initial
#         args array passed to the main method.
#
#         For COBOL applications, a single argument with a maximum
#         length of 100 characters can be used.
#---------------------------------------------------------------------*
#bcd.applicationArgs.1=Java argument element 1
#bcd.applicationArgs.2=Java argument element 2
#bcd.applicationArgs.3=Java argument element 3
#
#bcd.applicationArgs.1=COBOL single argument up to 100 characters
#
# 4
#---------------------------------------------------------------------*
# REQUIRED UPDATE: Support class names used to manage transactions.
#
#         For the DB2 JDBC driver, replace with the correct statement
#         for your installation.
#         If your application uses DB2 for z/OS, you MUST uncomment
#         this statement.
#
#         NOTE: A bcd.supportClass.1=class_name must be specified.
#         If you use the one provided by DB2, the DB2-related .jar
#         files and executables must be on the CLASSPATH and LIBPATH,
#         respectively.
#---------------------------------------------------------------------*
#bcd.supportClass.1=com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport
#
# 5
#---------------------------------------------------------------------*
# UPDATE: Verbose mode for additional diagnostics (default is false).
#---------------------------------------------------------------------*
#bcd.verbose=true
#
//
```

Figure 3. Example: JCL BCDIN configuration options (Part 2 of 2)

Procedure for modifying the BCDIN JCL

The following list summarizes the `BCDIN` JCL statements to use for configuring the Batch Runtime options:

__ **1** Specify the option that corresponds to the language of the application.

For example, in COBOL:

```
bcd.applicationLanguage=COBOL
```

For example, in Java:

```
bcd.applicationLanguage=JAVA
```

__ **2** Specify the program name or fully qualified Java class name of the application, where *MYPGMNAM* or *yourpackagename* is the name of the application.

For example, in COBOL:

```
bcd.applicationName=MYPGMNAM
```

For example, in Java:

```
bcd.applicationName=com.xyz.yourpackagename.classname
```

__ 3 Specify the program arguments that you want to pass to the program. Java and COBOL each have there own format.

For Java applications, you can use any number of arguments. Each argument is passed as an element of the initial arguments array passed to the main method. For example:

```
bcd.applicationArgs.1=java arg1
bcd.applicationArgs.2=java arg2
bcd.applicationArgs.3=java arg3
```

For COBOL applications, you can use a single argument with a maximum length of 100 characters. For example:

```
bcd.applicationArgs.1=COBOL single argument up to 100 characters
```

The *COBOL single argument...* value corresponds to the *PARM='string <=100 chars'* value of an `//EXECPGM EXEC PGM=Cobol_Main,PARM= JCL` statement.

__ 4 Specify the name of the support class used to manage the transaction. For example, the following statement for the DB2 JDBC driver should be uncommented from Figure 3 on page 12.

```
bcd.supportClass.1=com.ibm.db2.jcc.t2zos.T2zosBatchRuntimeSupport
```

__ 5 Specify the verbose mode, using *true* or *false*.

If you want diagnostic information, use the following statement:

```
bcd.verbose=true
```

If you do not want verbose mode, use the following:

```
bcd.verbose=false
```

Sample BCDPROC to invoke z/OS Batch Runtime

Figure 3 on page 12 shows an example of a BCDPROC statement. You can use a symbolic to override defaults on BCDPROC.

VERSION Specifies the Java SDK version (default 61).

LOGLVL Specifies the following JZOS trace level:

+I informational (default)

+T detail trace (used for additional diagnostics and debugging `//STDENV` script)

LEPARM Allows for additional Language Environment options to be specified by providing by a `//CEEDOPTS DD` statement. For more information, see *z/OS Language Environment Programming Reference* .

Note: z/OS Batch Runtime only supports EBCDIC file encoding.

```
//BCDPROC PROC VERSION='61',      JVMLDM version: 61 (Java 6.0.1 31bit)
//             LOGLVL='+I',       Debug level: +I(info) +T(trc)
//             LEPARM=''          Language Environment parms
//*
//*********************************************************************
//*                                                                   *
//*  Proprietary Statement:                                           *
//*                                                                   *
//*    Licensed Materials - Property of IBM                           *
//*    5694-A01                                                       *
//*    Copyright IBM Corp. 2011.                                      *
//*                                                                   *
//*    Status = HBB7780                                               *
//*                                                                   *
//*  Component = z/OS Batch Runtime (SC1BC)                           *
//*                                                                   *
//*  EXTERNAL CLASSIFICATION = OTHER                                  *
//*  END OF EXTERNAL CLASSIFICATION:                                  *
//*                                                                   *
//*  Sample procedure JCL to invoke z/OS Batch Runtime                *
//*                                                                   *
//*  Notes:                                                           *
//*                                                                   *
//*  1. Override the VERSION symbolic parameter in your JCL           *
//*     to match the level of the Java SDK you are running.           *
//*                                                                   *
//*       VERSION=61    Java SDK 6.0.1 (31 bit)                       *
//*                                                                   *
//*  2. Override the LOGLVL symbolic parameter to control             *
//*     the messages issued by the jZOS Java launcher.                *
//*                                                                   *
//*     Use the +T option when reporting problems to IBM or           *
//*     to diagnose problems in the STDENV script.                    *
//*                                                                   *
//*  3. Override the LEPARM symbolic parameter to add any             *
//*     application specific language environment options            *
//*     needed.                                                       *
//*                                                                   *
//*  Change History =                                                 *
//*                                                                   *
//*    $L0=BATCH,HBB7780,100324,KDKJ:                                 *
//*                                                                   *
//*                                                                   *
//*********************************************************************
//JAVA EXEC PGM=JVMLDM&VERSION,REGION=0M,
//             PARM='&LEPARM/&LOGLVL'
//*
//SYSPRINT  DD  SYSOUT=*           System stdout
//SYSOUT    DD  SYSOUT=*           System stderr
//STDOUT    DD  SYSOUT=*           Java System.out
//STDERR    DD  SYSOUT=*           Java System.err
//BCDOUT    DD  SYSOUT=*           Batch container messages
//BCDTRACE  DD  SYSOUT=*           Batch container trace
//*
//CEEDUMP   DD  SYSOUT=*
//*
```

Figure 4. Example: BCDPROC statement

Chapter 3. Defining connectivity for the database

This chapter describes basic information about setting up the z/OS Batch Runtime environment with the DB2 database and how the processing of transactions works for requests from COBOL or Java applications.

Considerations for setting up z/OS Batch Runtime services for a database resource

For the DB2 or database resource that z/OS Batch Runtime uses to make connections for interoperability functions, the database must do the following:

- Initialize the z/OS Batch Runtime environment processing
- End the z/OS Batch Runtime environment processing
- Obtain notification of the start of a global transaction
- Obtain notification of the completion of a global transaction.

DB2 Java Database Connectivity (JDBC) and z/OS Batch Runtime

At startup, the z/OS Batch Runtime calls the Java Database Connectivity (JDBC) driver for DB2 to establish a connection that can then be shared by the COBOL or Java applications. The DB2 JDBC detects the mode of z/OS Batch Runtime and creates the single physical attachment for processing applications. JDBC maintains this application attachment for any connection requests that an application makes. The COBOL and Java applications use the same "BatchRuntime" attachment to access the DB2 resources.

Establishing a connection to DB2 from a COBOL application usually requires three calls to the RRS Attach Facility (RRSAF):

- IDENTIFY
- SIGNON
- CREATE THREAD

Because the JDBC has created the DB2 resource attachment for the thread during z/OS Batch Runtime initialization, the COBOL application must not code these RRSAF calls to initialize or end a DB2 connection; otherwise, RRSAF fails the request. z/OS Batch Runtime performs resource clean up after processing ends for the request.

Transaction management and global transactions

z/OS Batch Runtime performs basic transaction management functions for the application through the Java Transaction API (JTA). It can manage the COBOL or Java application clients and can coordinate transaction management between itself and the z/OS RRS transaction management services.

All transactions that run on z/OS Batch Runtime are considered global transactions. z/OS Batch Runtime calls z/OS RRS to start a transaction to associate the transaction with the calling thread before it invokes the COBOL or Java application. The JDBC provides the following methods to perform transaction synchronization:

beforeCompletion	Invoked before the transaction process starts
afterCompletion	Invoked after the transaction is performed

The JDBC informs all of the active connections about the DB2 commit or rollback events for consistency in processing database requests. You cannot initiate DB2 commit or rollback requests from the COBOL or Java applications themselves. For this release, support for multiple resource managers is not available in z/OS Batch Runtime.

Commit and rollback services of z/OS Batch Runtime

COBOL invokes Batch Runtime methods for commit and rollback. For COBOL applications, z/OS Batch Runtime offers callable procedures for commit and rollback of a transaction. Before committing the unit of work, z/OS Batch Runtime invokes the beforeCompletion method on the JDBC to indicate the start of the commit. (This in turn invokes the z/OS RRS Commit_UR service to commit the transaction.) After the commit transaction is committed, z/OS Batch Runtime invokes the afterCompletion method on the JDBC to indicate the completion of the commit.

Before processing the rollback transaction, z/OS Batch Runtime invokes the beforeCompletion method on the JDBC to indicate the start of the rollback. (This in turn invokes the z/OS RRS Backout_UR service to back out the transaction.) After the rollback transaction is completed, z/OS Batch Runtime invokes the afterCompletion method on the JDBC to indicate completion of the rollback

End-of-job clean up processing

If the applications complete with no issues, z/OS Batch Runtime commits any outstanding transaction. z/OS Batch Runtime invokes the z/OS RRS end_transaction service to clean up a global transaction. It rolls back any outstanding global transaction and invokes the z/OS Resource Recovery Services (RRS) end_transaction service to pass a rollback action. It also communicates the start and completion of the transaction rollback process.

For additional information about RRS, see *z/OS MVS Programming: Resource Recovery* for topics about:

- Using Resource Recovery Services
- Callable Resource Recovery Services.

Chapter 4. Application interfaces for z/OS Batch Runtime

This section describes the following interfaces, considerations, and samples for z/OS Batch Runtime:

- Configuration options. See “Configuration options reference.”
- Helper functions including commit and rollback in Java. See “Helper functions for z/OS Batch Runtime” on page 21.
- Support elements for JDBC and DB2 communications. See“Support elements for JDBC and DB2” on page 22.
- Java environment variables. See “Java environment variables for z/OS Batch Runtime” on page 5.
- Language Environment considerations and restrictions for COBOL and Java applications. See “Language Environment restrictions for z/OS Batch Runtime” on page 22.
- Completion codes. See “Completion codes for z/OS Batch Runtime” on page 23.
- Code examples. See “Example: Java code calling COBOL” on page 31.

Configuration options reference

You can control z/OS Batch Runtime by using configuration options that you specify on the `//BCDIN JCL` statement. This section provides reference information about the supported input parameters. These *keyword*=*value* pairs are prefixed with '`bcd`'. For a description of the JCL conventions to specify options, see “JCL for BCDIN configurations options” on page 11.

Configuration option types

As Table 3 shows, the syntax of a configuration option varies according to the following types.

Table 3. Configuration option types

Type	Description and Example	Default
Keyword	An option in *keyword*=*value* format. Values can contain embedded blanks. Trailing blanks are removed. `bcd.applicationLanguage=COBOL`	None
Stem	An option you use to specify multiple values for the option. A numeric suffix (the stem) is appended to the option name and indicates the value number. A stem suffix must be numeric. Values can be skipped and can appear in any order; however, z/OS Batch Runtime processes the stem values in their numeric order. `bcd.supportClass.1=any.class.name`	None

Configuration option names

The following options are read from the `//BCDIN` JCL file. The name, description, and example of the option are provided.

bcd.applicationLanguage=*language*

Names the language of the application to be launched, where *language* is either **COBOL** or **JAVA**.

Default

None; the statement is required and must be specified as COBOL or JAVA.

Example

```
bcd.applicationLanguage=JAVA
```

bcd.applicationName=*application-name*

Names the fully qualified Java class or COBOL program name of the application, where *application-name* is the name of the application.

For COBOL applications, *application-name* is a 1-8 character module name. The z/OS Batch Runtime uses the typical z/OS LNKLST/STEPLIB search order for locating the COBOL application.

For Java applications, *application-name* is the fully qualified class name. The z/OS Batch Runtime uses the CLASSPATH environment variable to locate the main() method of the specified classname.

Default

None

Example

```
bcd.applicationName=XMPCOBJX
```

bcd.applicationArgs.*n=argument*

Names an argument to be passed to the application, where *n=argument* specifies the suffix number of the argument position.

For Java applications, each argument is passed as an element of the argument array that is passed to the main method.

For COBOL applications, you can specify only one argument. The argument can contain a maximum of 100 characters and is passed using the same convention as the `PARM= keyword` of the `// EXEC` JCL statement.

Default

None

Example

```
bcd.applicationArgs.1=java arg1
```

bcd.supportClass.*n=support-class-name*

Names a support class to be used with z/OS Batch Runtime, where *n=support-class-name* specifies a suffix number that indicates the order in which the support class is invoked.

Default

None, but at least one support class is required.

Example

```
bcd.supportClass.1=com.ibm.db2.jcc.t2zos.T2zosBatchRuntimeSupport
```

Note: For DB2, the following support class is provided by the JDBC driver. To use it, you must uncomment the following statement provided in the sample BCDBATCH job.

```
com.ibm.db2.jcc.t2zos.T2zosBatchRuntimeSupport
```

bcd.verbose=*value*

Specifies the verbose mode for the batch runtime, where *value* is either TRUE or FALSE. z/OS Batch Runtime generates additional diagnostics when you specify TRUE for verbose mode and can affect performance.

Default
FALSE

Example
```
bcd.verbose=true
```

Program arguments

You can pass program arguments to the COBOL or Java main application from z/OS Batch Runtime.

For COBOL programs, you can pass a single argument in standard format as it is specified on the PARM= keyword of the `//EXEC` JCL statement. The following statement shows an example:

```
bcd.applicationArgs.1=This is PARM= main arg to Cobol
```

For Java programs, you can pass program arguments as a string array to the Java main method, as shown in the following example; Java main methods accept this as a variable length string array per the usual specified behavior:

```
bcd.applicationArgs.1=500
bcd.applicationArgs.2=string input 1
bcd.applicationArgs.3=My userid
```

You do not have to include a single quote (') in the string value you are passing. Also, note that trailing blanks are not supported in the string.

Helper functions for z/OS Batch Runtime

As part of the interoperability commit and rollback database functions for COBOL and Java applications, z/OS Batch Runtime provides helper functions to simplify the processing.

For Java, methods for commit and rollback functions are available with the following package:

```
com.ibm.batch.spi.UserControlledTransactionHelper
```

Java function for commit and rollback

The following class contains commit and rollback functions for Java applications:

```
com.ibm.batch.spi.UserControlledTransactionHelper
```

The class contains the following static methods that initiate the commit or rollback process:

Commit	`UserControlledTransactionHelper.commit()`
Rollback	`UserControlledTransactionHelper.rollback()`

z/OS Batch Runtime uses z/OS Resource Recovery Services (RRS) to manage the unit of work that is active across the Java and COBOL language boundary. All commits and rollbacks must be managed by z/OS Batch Runtime; your applications should not call commit and rollback directly. Rather, they should use helper functions to call the functions. When your Java application needs to perform a commit or rollback, you must call these helper functions to perform the function. For COBOL applications, you can use the COBOL INVOKE statement to invoke these helper methods.

Direct use of JTA (Java transaction API) by Java programs is not allowed. Also, any use of SQL COMMIT or ROLLBACK APIs by Java or COBOL will be rejected with `SQLSTATE = '2D521 SQL COMMIT or ROLLBACK are invalid in the current operating environment'`. As such, Java programs should avoid setting the JDBC autocommit connection option. See "Code examples" on page 31 for examples.

Support elements for JDBC and DB2

You can use a support element (also referred to as a support class) to allow z/OS Batch Runtime to interoperate with a database or other resource manager.

For this release, the only support element is one that manages the JDBC driver that communicates with DB2. The support element must implement the following interface:

```
com.ibm.zos.zbatch.runtime.support.BCDBatchRuntimeSupport
```

This interface defines the following Java methods:

initializeBatchRuntimeEnv(Properties props)
Initializes z/OS Batch Runtime environment. Startup options are passed in the properties object

terminateBatchRuntimeEnv()
Ends the z/OS Batch Runtime environment.

notifyNewGlobalTransaction(BCDTransaction transaction)
Informs the support element of a new global transaction. The support element of this method calls the following, which z/OS Batch Runtime implements:

```
transaction.registerSynchronization(BCDSynchronization sync)
```

getVersion()
Retrieves a string representation of the version of the support element for diagnostic purposes. The content of the string is determined by the support element.

Transaction and synchronization processing that are normally part of the J2EE javax.transaction package are part of the following z/OS Batch Runtime package:

```
com.ibm.zos.batch.runtime.support.transaction
```

The classes for this package are called:

- BCDTransaction
- BCDSynchronization

In addition, a support element is required to implement a static getInstance() method that returns an instance of the support element class. You must add any .JAR files or DLLs to the CLASSPATH and LIBPATH in the BCDBATCH JCL. For more details, see Chapter 2, "Invoking z/OS Batch Runtime," on page 5.

Language Environment restrictions for z/OS Batch Runtime

Certain restrictions apply to COBOL and Java applications that use the Language Environment in the z/OS Batch Runtime environment.

- COBOL applications must not use the STOP RUN statement. Using the option in COBOL programs prevents the z/OS Batch Runtime environment from receiving control. Instead, use the GOBACK statement to end the COBOL application.

- COBOL will no longer be the first program entered. COBOL-specific runtime options might be affected.
- Java applications must be single threaded and must not use the *system.exit* method. Using the *system.exit* method ends the JVM and prevents the z/OS Batch Runtime environment from receiving control. Instead, end the main Java procedure with a simple return statement.

Completion codes for z/OS Batch Runtime

Upon completion, z/OS Batch Runtime processing returns the condition codes shown in Table 4.

Table 4. Completion codes for z/OS Batch Runtime

Code	Description
00	The processing has successfully completed.
08	The processing has launched the application, but the application has returned a non-zero condition code. See the z/OS Batch Runtime messages in the job log for errors.
12	An error occurred during z/OS Batch Runtime processing. See the z/OS Batch Runtime messages in the job log for errors and *z/OS MVS System Messages, Vol 3 (ASB-BPX)* for more information.

Chapter 5. Application structure and build considerations

The following sections describe considerations for how to structure and build your applications when using z/OS Batch Runtime.

DLL considerations for COBOL and Java

In effort to simplify, some information from *Enterprise COBOL for z/OS, V4R2, Programming Guide, SC23-8529* is repeated in this section of the documentation. For complete details, see *Enterprise COBOL for z/OS, V4R2, Programming Guide* at `http://publibfp.boulder.ibm.com/epubs/pdf/igy3pg50.pdf`

It is important to recognize the structural implications to COBOL source files when they are calling out to Java. In particular, you need DLL and RECURSIVE on COBOL classes and methods or on COBOL programs that invoke Java methods.

To compile COBOL source code that contains OO syntax, such as INVOKE statements or class definitions, or that use Java services, you must use these compiler options:

- RENT
- DLL
- THREAD

Any programs that you compile with the THREAD compiler option must be recursive. You must specify the recursive clause in the PROGRAM-ID paragraph of each OO COBOL client program. This can affect the overall COBOL program structure because a program compiled with a DLL cannot make a traditional COBOL dynamic call. It can, however, be statically linked with and call into another COBOL program compiled dynamic. This separate but statically linked program can then use a traditional dynamic call to other external COBOL modules with built dynamic programs.

In general, DLL linkage built COBOL programs can only call out to other external DLL linkage built programs. Similarly, dynamic call built COBOL programs can only call out to other external dynamic call built programs. However, static linking of objects with either two of these external program call mechanisms is allowed. This provides the bridging between the DLL linkage that Java requires and the traditional COBOL dynamic call.

For additional details, see the topic about "Using DLL linkage and dynamic calls together " in *Enterprise COBOL for z/OS, V4R2, Programming Guide*.

Example of a COBOL COMMIT wrapper

Figure 5 on page 26 is a simple example of a COBOL COMMIT wrapper that, while compiled with the DLL option required for Java, can be statically linked with a main non-DLL application module. In this example, a procedural COBOL program invokes a Java method. Only the non-DLL module objects that need to call the new COMMIT function need to be recompiled. You can also perform a similar function for ROLLBACK.

```
*-----------------------------------------------------------------
*
* Program Name   : COBCOMIT
* Objective      : Call RSS global commit via batch container
*
*-----------------------------------------------------------------
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "COBCOMIT" IS RECURSIVE.

/
 ENVIRONMENT DIVISION.
*---------------------
 CONFIGURATION SECTION.
 REPOSITORY.
     Class JavaException is "java.lang.Exception"
     Class UserControlledTransaction is
           "com.ibm.batch.spi.UserControlledTransactionHelper".
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.

 DATA DIVISION.
 FILE SECTION.

 WORKING-STORAGE SECTION.
 01 ex object reference JavaException.

 LINKAGE SECTION.
 01 RETCODE PIC S9(9) USAGE IS BINARY.
 COPY JNI.

 PROCEDURE DIVISION RETURNING RETCODE.
*-----------------------------------------------------------------
*
*    Test batch cobol commit.
*
*-----------------------------------------------------------------
 PROGRAM-BEGIN.
     SET ADDRESS OF JNIENV TO JNIENVPTR
     SET ADDRESS OF JNINATIVEINTERFACE TO JNIENV
     Display "Calling into Java commit"
     Invoke UserControlledTransaction "commit"
     Display "Returned from Java commit"
     Perform ErrorCheck
     Goback
     .
 PROGRAM-END.
     GOBACK.
* need to perform exception check / stack trace at this point ?

ErrorCheck.
    Compute RETCODE = 0
    Call ExceptionOccurred
         using by value JNIEnvPtr
         returning ex
    If ex not = null then
         Call ExceptionClear using by value JNIEnvPtr
         Display "Caught an unexpected exception"
         Invoke ex "printStackTrace"
         Compute RETCODE = 99
    End-if
     .
  End program "COBCOMIT".
```

Figure 5. Example: COBOL COMMIT wrapper

Figure 6 shows an example of the JCL that would be needed to compile the COMMIT wrapper shown in Figure 5 on page 26.

```
//COMPCMIT JOB ,'STEVE PROGRAM ',
//             NOTIFY=&SYSUID,
//             MSGCLASS=X,
//             CLASS=A,
//             REGION=0M,
//             TIME=120
//COMPILE  EXEC IGYWC,LNGPRFX='SYSPROG.MNT.COBOL42',
//         COND=(4,LT),
//         PARM.COBOL=(NOSEQUENCE,RENT,LIB,THREAD,
//         NODYNAM,DLL)
//COBOL.SYSLIB DD DSN=SUIMGJB.PRIVATE.JNI.COPY,
//             DISP=SHR
//COBOL.SYSIN  DD DSN=SUIMGJB.PRIVATE.DOCXMP.COBOL(COBCOMIT),
//             DISP=SHR
//COBOL.SYSLIN DD DSN=SUIMGJB.PRIVATE.COBOL.OBJ(COBCOMIT),
//             DISP=SHR
//
```

Figure 6. Example: JCL used to compile COMMIT wrapper

Examples of program structures

This section demonstrates several types of program structures and interaction between COBOL, Java, and z/OS Batch Runtime.

Figure 7 on page 28 shows an overview of a COBOL program that interacts with a Java program. In this example, the program flow starts in COBOL and then flows to a Java program and to another COBOL program. OOCOBOL methods are not used; however, the programs use both COBOL JNI and user JNI.

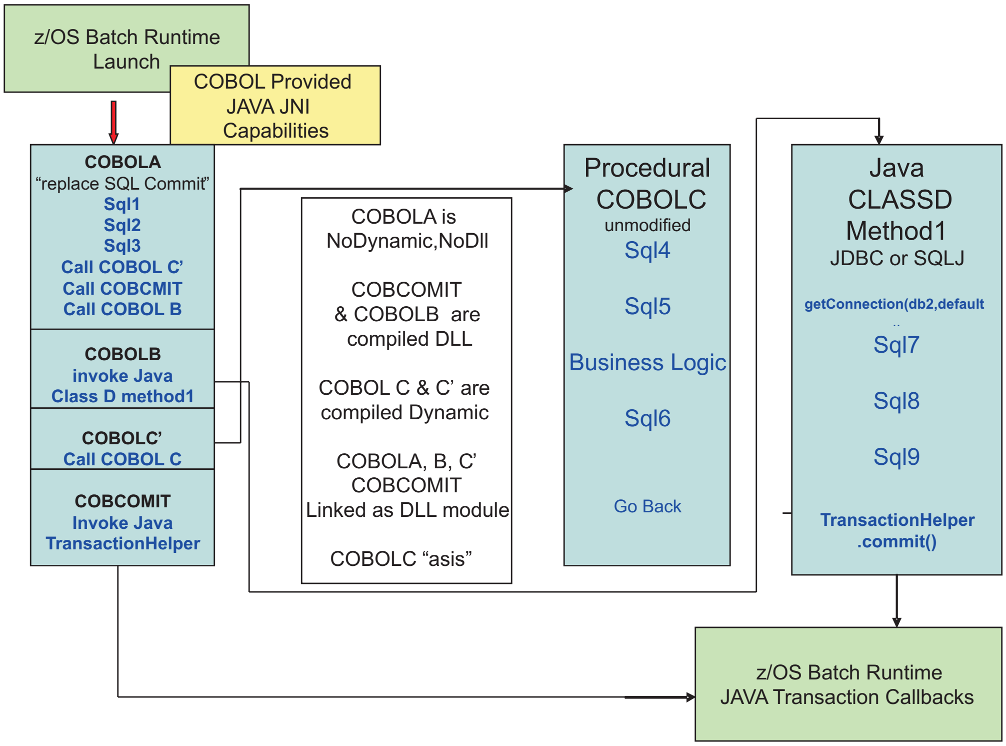

z/OS Batch Runtime Launch
COBOL Provided JAVA JNI Capabilities
COBOLA
“replace SQL Commit”
Sql1
Sql2
Sql3
Call COBOL C’
Call COBCMIT
Call COBOL B
COBOLB
invoke Java Class D method1
COBOLC’
Call COBOL C
COBCOMIT
Invoke Java TransactionHelper
COBOLA is NoDynamic,NoDll
COBCOMIT & COBOLB are compiled DLL
COBOL C & C’ are compiled Dynamic
COBOLA, B, C’ COBCOMIT Linked as DLL module
COBOLC “asis”
Procedural COBOLC
unmodified
Sql4
Sql5
Business Logic
Sql6
Go Back
Java CLASSD Method1
JDBC or SQLJ
getConnection(db2,default
..
Sql7
Sql8
Sql9
TransactionHelper .commit()
z/OS Batch Runtime JAVA Transaction Callbacks

Figure 7. Example: COBOL program calling Java and unmodified COBOL

In Figure 8, a Java program flows to a COBOL program. In this example, the Java program uses an OOCOBOL factory wrapper to call COBOL.

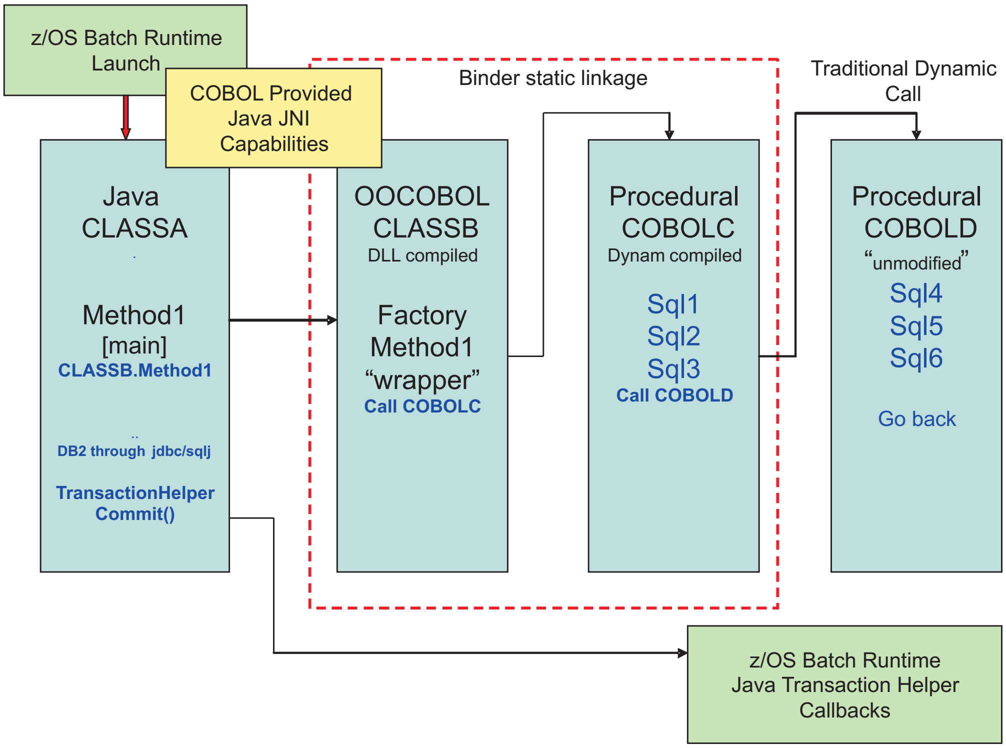


Figure 8. Example: Java program using OOCOBOL to call COBOL

Building programs: compile and link JCL examples

For complete documentation about building COBOL applications, including Object Oriented (OO) COBOL, see *Enterprise COBOL for z/OS, V4R2, Programming Guide*.

For compiling with JCL, IBM provides a set of cataloged procedures to reduce the amount of JCL coding that you need to write. If the cataloged procedures do not meet your needs, you can write your own JCL. Using JCL, you can compile a single program or compile several programs as part of a batch job. See Chapter 2, “Invoking z/OS Batch Runtime,” on page 5 for more information.

The compiler translates your COBOL program into language that the computer can process (object code). The compiler also lists errors in your source statements and provides supplementary information to help you debug and tune your program. Use compiler-directing statements and compiler options to control your compilation. After compiling your program, you need to review the results of the compilation and correct any compiler-detected errors.

To build Java programs, use the `javac` command to create the classes and the `jar` command for packaging. This documentation focuses on building a typical use case that updates a traditional COBOL program to call out to Java methods in which either or both can use DB2.

The JCL example shown in Figure 9 on page 30 is a modification of a sample COBOL DB2 phone program that ships as part of IBM DB2 for z/OS. This program is typically found in *hlq*.sdsnsamp(DSN8BC3) and is often used in the DB2 installation verification program (IVP). The COBOL source is modified to invoke a simple Java "Hello World" method that also selects rows from the DB2 catalog using the SYSIBM schema. The following are implications on the DB2 provided COBOL build procedure to run in the z/OS Batch Runtime container:

- The Language Environment Runtime library CEE.SCEERUN must be in the JOBLIB for the Java JNI support.
- The ATTACH(RRSAF) must be in the preprocessor portion of the catalogued procedure. Although optional, this forces the generation of RRS attach entry point at compile time. Omit this option for attach-neutral code generation. The z/OS Batch Runtime requires the use of RRS attach to be bound at compile (as in this example), link (include `DSNRLI`), or runtime (include `DSNULI`).
- The use of Java from COBOL source requires the compile options `RENT,DLL,THREAD`.
- The long names required for the Java JNI imply use of PDSE libraries by the binder (rather than traditional PDS load libraries).
- The input to the Binder requires both the Enterprise COBOL Java linkage and JNI export (*.x) files.

```
//COBBUILD   JOB (MOP,1458),'STEVE',CLASS=A,REGION=0M,
//     MSGLEVEL=(1,1),MSGCLASS=X,TIME=1440,NOTIFY=&SYSUID
//*
//**********************************************************************
//*  NAME = DSNTEJ2C -- MODIFIED FOR RRS AND Java BATCH CONTAINER RUN  *
//*                     BUILD ONLY WITH APP CALL TO JAVA               *
//*  DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION W CALL TO JAVA          *
//*                     PHASE 2                                        *
//*                     COBOL                                          *
//*                                                                    *
//*    LICENSED MATERIALS - PROPERTY OF IBM                            *
//*    5635-DB2                                                        *
//*    (C) COPYRIGHT 1982, 2006 IBM CORP.  ALL RIGHTS RESERVED.        *
//*                                                                    *
//*    STATUS = VERSION 9                                              *
//*                                                                    *
//*  FUNCTION = THIS JCL PERFORMS THE PHASE 2 COBOL SETUP FOR THE      *
//*             SAMPLE APPLICATIONS.  IT PREPARES AND EXECUTES         *
//*             COBOL BATCH PROGRAMS.                                  *
//*                                                                    *
//*             THIS JOB IS RUN AFTER PHASE 1.                         *
//*                                                                    *
//*                                                                    *
//* CHANGE ACTIVITY =                                                  *
//*                                                                    *
//**********************************************************************
//JOBLIB  DD  DISP=SHR,DSN=DSN910.NEWFUNC.SDSNEXIT
//        DD  DISP=SHR,DSN=DSN910.SDSNLOAD
//        DD  DISP=SHR,DSN=CEE.SCEERUN
//*
//*        PREPARE COBOL PHONE PROGRAM
//PH02CS03 EXEC DSNHNCOB,MEM=XMPCOBJV,
//         COND=(4,LT),
//         PARM.PC=('HOST(IBMCOB)',APOST,APOSTSQL,SOURCE,
//         NOXREF,'SQL(DB2)','DEC(31)','ATTACH(RRSAF)'),
//         PARM.COB=(NOSEQUENCE,LIB,QUOTE,RENT,'PGMNAME(LONGUPPER)',
//         DLL,THREAD)
//PC.DBRMLIB   DD DSN=DSN910.DBRMLIB.DATA(XMPCOBJV),
//             DISP=SHR
//PC.SYSLIB    DD DSN=SUIMGJB.PRIVATE.DSN910.SRCLIB.DATA,
//             DISP=SHR
//PC.SYSIN     DD DSN=SUIMGJB.PRIVATE.JCL.CNTL(XMPCOBJV),
//             DISP=SHR
//COB.SYSLIB   DD DSN=SUIMGJB.PRIVATE.JNI.COPY,
//             DISP=SHR
//LKED.SYSLMOD DD DSN=SUIMGJB.PRIVATE.LIBRARY(XMPCOBJV),
//             DISP=SHR
//LKED.RUNLIB  DD DSN=DSN910.RUNLIB.LOAD,
//             DISP=SHR
//LKED.SYSIN   DD *
     INCLUDE SYSLIB(DSNRLI)
     INCLUDE RUNLIB(DSN8MCG)
     INCLUDE '/home/cob42/cobol/lib/igzcjava.x'
     INCLUDE '/usr/lpp/java/J6.0/lib/s390/j9vm/libjvm.x'
//
```

Figure 9. Example: JCL for COBOL DB2 phone program

Code examples

This section contains the following code examples:

- “Example: Java code calling COBOL”
- “Example: C DLL calling COBOL from Java” on page 33
- “Example: COBOL code invoking Java” on page 33

Example: Java code calling COBOL

Figure 10 shows an example of Java code calling COBOL.

```
package com.ibm.zos.batch.container.test;

import java.sql.*;
import com.ibm.batch.spi.UserControlledTransactionHelper;
import com.ibm.ws.gridcontainer.exceptions.TransactionManagementException;

public class Sample
{
  //Native method declaration
  private native int CallCOBOL();
  //Load the library
  static {
  System.loadLibrary("c_to_cobol");
}

public static void main(String[] args)
{
Connection conn = DriverManager.getConnection(url);
  String url = "jdbc:default:connection";

  Statement stmt;
  int maxRows = 25;
  String pnumber = "";
  int pnum = 0;
  int rc = 0;
  String formatted;

  try
    {
      System.out.println ( "Establishing Connection to URL: " + url );

      conn = DriverManager.getConnection(url);
      System.out.println ( " successful connect" );
      stmt = conn.createStatement();
      System.out.println ( " Successful creation of Statement" );
      // Limit the number of rows to return
      stmt.setMaxRows ( maxRows );
```

Figure 10. Example: Java code calling COBOL (Part 1 of 2)

```
  // SELECT from an DB2 sample table
  String sqlText =
  "SELECT PHONENUMBER " +
  "FROM DSN8910.VEMPLP " +
  "WHERE EMPLOYEENUMBER = '000260'";
  ResultSet results = stmt.executeQuery ( sqlText );
  pnumber = results.getString ( "PHONENUMBER" );
  pnum = Integer.parseInt(pnumber.trim());
  pnum++;
  pnum = pnum % 10000;
  formatted = String.format("%04d", pnum);

  sqlText =
  "UPDATE DSN8910.VEMPLP " +
  " SET PHONENUMBER = " + "'"+formatted+"'" +
  " WHERE EMPLOYEENUMBER = '000260' ";
  int updateCount = stmt.executeUpdate(sqlText);
  System.out.println ( "Successful execution of UPDATE. Rows updated= " + updateCount );

  // close ResultSet and Statement
  results.close();
  // Call COBOL via a C DLL
  Sample call_cobol = new Sample();
  //Call native method
  rc = call_cobol.CallCOBOL();
  System.out.println ( "Returned from COBOL with a rc: " + rc );

  if (rc == 0)
   {
    try
    {
     UserControlledTransactionHelper.commit();
    }
    catch (TransactionManagementException e)
    {
     e.printStackTrace();
    }
   }
  else
   {
    try
    {

    UserControlledTransactionHelper.rollback();
    }
    catch (TransactionManagementException e)
    {
     e.printStackTrace();
    }
   }
  }
  catch (SQLException ex)
   {
    System.out.println("SQLException information");
    while(ex!=null) {
    System.err.println ("Error msg: " + ex.getMessage());
    System.err.println ("SQLSTATE: " + ex.getSQLState());
    System.err.println ("Error code: " + ex.getErrorCode());
    ex.printStackTrace();
    ex = ex.getNextException();
   }
  }
 }
}
```

Figure 10. Example: Java code calling COBOL (Part 2 of 2)

Example: C DLL calling COBOL from Java

The example in Figure 11 shows the C interface DLL to use when calling COBOL.

```
/ c99  -o libc_to_cobol.so -Wc,exportall -Wl,
dll -I/usr/lpp/java/J6.0.1/include
    -I/usr/lpp/java/J6.0.1/include/zos c_to_cobol.c

#include <jni.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include "com_ibm_zos_batch_container_test_Java_Calls_Cobol.h"

void (*fetch(const char *name))();
typedef void cfunc();

JNIEXPORT jint JNICALL
Java_com_ibm_zos_batch_container_test_Java_1Calls_1Cobol_CallCOBOL(JNIEnv * jenv, jobject jobj)
{
 cfunc *cobfetch_ptr;
 cobfetch_ptr = (cfunc *) fetch("XMPCOBJ3");    // loads fetched module
 if (cobfetch_ptr == NULL){
     printf("\tfetch failed\n");
 }
 else
 {
    printf("\tShould be going off to COBOL\n\n");
    (*cobfetch_ptr)();         // sets up the proper linkage for the call
 }

 return(0);
}
```

Figure 11. Example: C interface DLL for calling COBOL from Java

Example: COBOL code invoking Java

Figure 12 on page 34 is an example of a modified DB2 sample phone application that uses COBOL code to invoke the "sayHello" Java method. Descriptions for each of the code blocks precede the example.

Figure 12 on page 34 includes changes that were made in the sample program to provide an interface to Java. These changes are **highlighted** and are located in the following areas of the example:

A Identification Division

B Environment Division

C Linkage Section

D Main Program Routine

E Updates Phone Numbers For Employees

F Perform Rollback

G Java Exception Check

Note: This sample was provided by DB2 , typically in *hlq*.sdsnsamp(DSN8BC3). For more details, see http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc/db2prodhome.htm

```
   IDENTIFICATION DIVISION.
   *-----------------------
A  PROGRAM-ID. DSN8BC3 RECURSIVE.

   ****** DSN8BC3 - DB2 SAMPLE PHONE APPLICATION - COBOL - BATCH ***
   * *
   * MODULE NAME = DSN8BC3 *
   * *
   * DESCRIPTIVE NAME = DB2 SAMPLE APPLICATION *
   * PHONE APPLICATION *
   * BATCH *
   * COBOL *
   * *
   *LICENSED MATERIALS - PROPERTY OF IBM *
   *5695-DB2 *
   *(C) COPYRIGHT 1982, 1995 IBM CORP. ALL RIGHTS RESERVED. *
   * * *---------------------------------------------------------------*
   /

   ENVIRONMENT DIVISION.
   *--------------------
   CONFIGURATION SECTION.
   SPECIAL-NAMES. C01 IS TO-TOP-OF-PAGE.
   REPOSITORY.
B  Class HelloJ is
   "com.ibm.zos.batch.container.test.HelloJ"
   Class JavaException is "java.lang.Exception"
   Class BCDTranHelper is
   "com.ibm.batch.spi.UserControlledTransactionHelper".
   INPUT-OUTPUT SECTION.
   FILE-CONTROL.
   SELECT CARDIN
   ASSIGN TO DA-S-CARDIN.
   SELECT REPOUT
   ASSIGN TO UT-S-REPORT.

   DATA DIVISION.
   *-------------
   FILE SECTION.
   FD CARDIN
   RECORD CONTAINS 80 CHARACTERS
   BLOCK CONTAINS 0 RECORDS
   LABEL RECORDS ARE OMITTED.
   01 CARDREC PIC X(80).

   FD REPOUT
   RECORD CONTAINS 120 CHARACTERS
   LABEL RECORDS ARE OMITTED
   DATA RECORD IS REPREC.
   01 REPREC PIC X(120).
```

Figure 12. Example: COBOL DB2 phone application that invokes Java under z/OS Batch Runtime (Part 1 of 10)

```
/
WORKING-STORAGE SECTION.

*****************************************************
* STRUCTURE FOR INPUT *
*****************************************************
01 IOAREA.
02 ACTION PIC X(01).
02 LNAME PIC X(15).
02 FNAME PIC X(12).
02 ENO PIC X(06).
02 NEWNO PIC X(04).
02 FILLER PIC X(42).

01 ex object reference JavaException.
*****************************************************
* REPORT HEADER STRUCTURE *
*****************************************************
01 REPHDR1.
02 FILLER PIC X(29)
VALUE '-----------------------------'.
02 FILLER PIC X(21)
VALUE ' TELEPHONE DIRECTORY '.
02 FILLER PIC X(29)
VALUE '-----------------------------'.
01 REPHDR2.
02 FILLER PIC X(09) VALUE 'LAST NAME'.
02 FILLER PIC X(07) VALUE SPACES.
02 FILLER PIC X(10) VALUE 'FIRST NAME'.
02 FILLER PIC X(03) VALUE SPACES.
02 FILLER PIC X(08) VALUE 'INITIAL'.
02 FILLER PIC X(07) VALUE 'PHONE'.
02 FILLER PIC X(09) VALUE 'EMPLOYEE'.
02 FILLER PIC X(05) VALUE 'WORK'.
02 FILLER PIC X(04) VALUE 'WORK'.
01 REPHDR3.
02 FILLER PIC X(37) VALUE SPACES.
02 FILLER PIC X(07) VALUE 'NUMBER'.
02 FILLER PIC X(09) VALUE 'NUMBER'.
02 FILLER PIC X(05) VALUE 'DEPT'.
02 FILLER PIC X(05) VALUE 'DEPT'.
02 FILLER PIC X(04) VALUE 'NAME'.

*****************************************************
* REPORT STRUCTURE *
*****************************************************
01 REPDATA.
02 RLNAME PIC X(15).
02 FILLER PIC X(01) VALUE SPACES.
02 RFNAME PIC X(12).
02 FILLER PIC X(04) VALUE SPACES.
02 RMIDINIT PIC X(01).
02 FILLER PIC X(04) VALUE SPACES.
02 RPHONE PIC X(04).
02 FILLER PIC X(03) VALUE SPACES.
02 REMPNO PIC X(06).
02 FILLER PIC X(03) VALUE SPACES.
02 RDEPTNO PIC X(03).
02 FILLER PIC X(02) VALUE SPACES.
02 RDEPTNAME PIC X(36).
```

Figure 12. Example: COBOL DB2 phone application that invokes Java under z/OS Batch Runtime (Part 2 of 10)

```
******************************************************
* WORKAREAS *
******************************************************
01 LNAME-WORK.
49 LNAME-WORKL PIC S9(4) COMP.
49 LNAME-WORKC PIC X(15).
01 FNAME-WORK.
49 FNAME-WORKL PIC S9(4) COMP.
49 FNAME-WORKC PIC X(12).
77 INPUT-SWITCH PIC X VALUE 'Y'.
88 NOMORE-INPUT VALUE 'N'.
77 NOT-FOUND PIC S9(9) COMP VALUE +100.
******************************************************
* VARIABLES FOR ERROR-HANDLING *
******************************************************
01 ERROR-MESSAGE.
02 ERROR-LEN PIC S9(4) COMP VALUE +960.
02 ERROR-TEXT PIC X(120) OCCURS 10 TIMES
INDEXED BY ERROR-INDEX.
77 ERROR-TEXT-LEN PIC S9(9) COMP VALUE +120.

77 W09-WAIT-TIME PIC S9(8) COMP VALUE 0005.
77 W09-RESPONSE PIC S9(8) COMP VALUE 0000.

******************************************************
* SQL INCLUDE FOR SQLCA *
******************************************************
EXEC SQL INCLUDE SQLCA END-EXEC.

******************************************************
* SQL DECLARATION FOR VIEW VPHONE *
******************************************************
EXEC SQL DECLARE DSN8910.VPHONE TABLE
(LASTNAME VARCHAR(15) NOT NULL,
FIRSTNAME VARCHAR(12) NOT NULL,
MIDDLEINITIAL CHAR(01) NOT NULL,
PHONENUMBER CHAR(04) ,
EMPLOYEENUMBER CHAR(06) NOT NULL,
DEPTNUMBER CHAR(03) NOT NULL,
DEPTNAME VARCHAR(36) NOT NULL)
END-EXEC.

******************************************************
* STRUCTURE FOR PHONE RECORD *
******************************************************
01 PPHONE.
02 LASTNAME.
49 LASTNAMEL PIC S9(4) COMP.
49 LASTNAMEC PIC X(15) VALUE SPACES.
02 FIRSTNAME.
49 FIRSTNAMEL PIC S9(4) COMP.
49 FIRSTNAMEC PIC X(12) VALUE SPACES.
02 MIDDLEINITIAL PIC X(01).
02 PHONENUMBER PIC X(04).
02 EMPLOYEENUMBER PIC X(06).
02 DEPTNUMBER PIC X(03).
02 DEPTNAME.
49 DEPTNAMEL PIC S9(4) COMP.
49 DEPTNAMEC PIC X(36) VALUE SPACES.
*
77 PERCENT-COUNTER PIC S9(4) COMP.
```

Figure 12. Example: COBOL DB2 phone application that invokes Java under z/OS Batch Runtime (Part 3 of 10)

```
    *****************************************************
    * SQL DECLARATION FOR VIEW VEMPLP *
    *****************************************************
    EXEC SQL DECLARE DSN8910.VEMPLP TABLE
    (EMPLOYEENUMBER CHAR(06) NOT NULL,
    PHONENUMBER CHAR(04) )
    END-EXEC.
    *****************************************************
    * SQL CURSORS *
    *****************************************************
    *** CURSOR LISTS ALL EMPLOYEE NAMES

    EXEC SQL DECLARE TELE1 CURSOR FOR
    SELECT *
    FROM DSN8910.VPHONE
    END-EXEC.

    *** CURSOR LISTS ALL EMPLOYEE NAMES WITH A PATTERN (%) OR (_)
    *** FOR LAST NAME

    EXEC SQL DECLARE TELE2 CURSOR FOR
    SELECT *
    FROM DSN8910.VPHONE
    WHERE LASTNAME LIKE :LNAME-WORK
    AND FIRSTNAME LIKE :FNAME-WORK
    END-EXEC.

    *** CURSOR LISTS ALL EMPLOYEES WITH A SPECIFIC
    *** LAST NAME

    EXEC SQL DECLARE TELE3 CURSOR FOR
    SELECT *
    FROM DSN8910.VPHONE
    WHERE LASTNAME = :LNAME
    AND FIRSTNAME LIKE :FNAME-WORK
    END-EXEC.
    /
    /****************************************************
    * FIELDS SENT TO MESSAGE ROUTINE *
    *****************************************************
    01 MAJOR PIC X(07) VALUE 'DSN8BC3'.

    01 MSGCODE PIC X(4).

    01 OUTMSG PIC X(69).

    01 MSG-REC1.
    02 OUTMSG1 PIC X(69).
    02 RETCODE PIC S9(9).

    01 MSG-REC2.
    02 OUTMSG2 PIC X(69).

C   LINKAGE SECTION.
    COPY JNI.

    PROCEDURE DIVISION.
    *------------------
```

Figure 12. Example: COBOL DB2 phone application that invokes Java under z/OS Batch Runtime (Part 4 of 10)

```
  *****************************************************
  * SQL RETURN CODE HANDLING *
  *****************************************************
  EXEC SQL WHENEVER SQLERROR GOTO DBERROR END-EXEC.
  EXEC SQL WHENEVER SQLWARNING GOTO DBERROR END-EXEC.
  EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.

  *****************************************************
  * MAIN PROGRAM ROUTINE *
  *****************************************************
  PROG-START.
  MOVE 0 to RETURN-CODE.
  SET ADDRESS OF JNIENV TO JNIENVPTR
  SET ADDRESS OF JNINATIVEINTERFACE TO JNIENV
D Invoke HelloJ "sayHello"
  Display "Returned from Java sayHello to MAIN"
  Perform ErrorCheck
  * **OPEN FILES
  OPEN INPUT CARDIN
  OUTPUT REPOUT.

  * **GET FIRST INPUT
  READ CARDIN RECORD INTO IOAREA
  AT END MOVE 'N' TO INPUT-SWITCH.

  * **MAIN ROUTINE
  PERFORM PROCESS-INPUT
  UNTIL NOMORE-INPUT.
  PROG-END.
  * **CLOSE FILES
  CLOSE CARDIN
  REPOUT.

  GOBACK.

  *****************************************************
  * CREATE REPORT HEADING *
  * SELECT ACTION *
  *****************************************************
  PROCESS-INPUT.
  * **PRINT HEADING
  WRITE REPREC FROM REPHDR1
  AFTER ADVANCING TO-TOP-OF-PAGE.
  WRITE REPREC FROM REPHDR2
  AFTER ADVANCING 2 LINES.
  WRITE REPREC FROM REPHDR3.

  * **SELECT ACTION
  IF ACTION = 'L'
  PERFORM LIST-FUNCTION
  ELSE
  IF ACTION = 'U'
  PERFORM TELEPHONE-UPDATE
```

Figure 12. Example: COBOL DB2 phone application that invokes Java under z/OS Batch Runtime (Part 5 of 10)

```
ELSE
* **INVALID REQUEST
* **PRINT ERROR MESSAGE
MOVE '068E' TO MSGCODE
CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
MOVE OUTMSG TO OUTMSG2
WRITE REPREC FROM MSG-REC2
AFTER ADVANCING 2 LINES.
READ CARDIN RECORD INTO IOAREA
AT END MOVE 'N' TO INPUT-SWITCH.
/
*****************************************************
* DETERMINE FORM OF NAME USED TO LIST EMPLOYEES *
*****************************************************
LIST-FUNCTION.
* **NO LAST NAME GIVEN
IF LNAME = SPACES
MOVE '%' TO LNAME.
* **NO FIRST NAME GIVEN
IF FNAME = SPACES
MOVE '%' TO FNAME.
* **LIST ALL EMPLOYEES
IF LNAME = '*'
PERFORM LIST-ALL
ELSE
* **UNSTRING LAST NAME
UNSTRING LNAME
DELIMITED BY SPACE
INTO LNAME-WORKC
COUNT IN LNAME-WORKL
* **UNSTRING FIRST NAME
UNSTRING FNAME
DELIMITED BY SPACE
INTO FNAME-WORKC
COUNT IN FNAME-WORKL
* **COUNT %'S
MOVE ZERO TO PERCENT-COUNTER
INSPECT LNAME
TALLYING PERCENT-COUNTER FOR ALL '%'
IF PERCENT-COUNTER > ZERO
* **IF NO %'S THEN
* **LIST SPECIFIC NAME(S)
* **ELSE
* **LIST GENERIC NAME(S)
PERFORM LIST-GENERIC
ELSE
PERFORM LIST-SPECIFIC.
/
*****************************************************
* LIST ALL EMPLOYEES *
*****************************************************
LIST-ALL.
* **OPEN CURSOR
EXEC SQL OPEN TELE1 END-EXEC
* **GET EMPLOYEES
EXEC SQL FETCH TELE1 INTO :PPHONE END-EXEC.
```

Figure 12. Example: COBOL DB2 phone application that invokes Java under z/OS Batch Runtime (Part 6 of 10)

```
IF SQLCODE = NOT-FOUND
* **NO EMPLOYEE FOUND
* **PRINT ERROR MESSAGE
MOVE '008I' TO MSGCODE
CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
MOVE OUTMSG TO OUTMSG2
WRITE REPREC FROM MSG-REC2
AFTER ADVANCING 2 LINES
ELSE
* **LIST ALL EMPLOYEES
PERFORM PRINT-AND-GET1
UNTIL SQLCODE IS NOT EQUAL TO ZERO.

* **CLOSE CURSOR
EXEC SQL CLOSE TELE1 END-EXEC.

PRINT-AND-GET1.
PERFORM PRINT-A-LINE.
EXEC SQL FETCH TELE1 INTO :PPHONE END-EXEC.
/
*****************************************************
* LIST GENERIC EMPLOYEES *
*****************************************************
LIST-GENERIC.
* **OPEN CURSOR
EXEC SQL OPEN TELE2 END-EXEC.

* **GET EMPLOYEES
EXEC SQL FETCH TELE2 INTO :PPHONE END-EXEC.

IF SQLCODE = NOT-FOUND
* **NO EMPLOYEE FOUND
* **PRINT ERROR MESSAGE
MOVE '008I' TO MSGCODE
CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
MOVE OUTMSG TO OUTMSG2
WRITE REPREC FROM MSG-REC2
AFTER ADVANCING 2 LINES
ELSE
* **LIST GENERIC EMPLOYEE(S)
PERFORM PRINT-AND-GET2
UNTIL SQLCODE IS NOT EQUAL TO ZERO.

* **CLOSE CURSOR
EXEC SQL CLOSE TELE2 END-EXEC.

PRINT-AND-GET2.
PERFORM PRINT-A-LINE.
EXEC SQL FETCH TELE2 INTO :PPHONE END-EXEC.
/
*****************************************************
* LIST SPECIFIC EMPLOYEES *
*****************************************************
LIST-SPECIFIC.
* **OPEN CURSOR
EXEC SQL OPEN TELE3 END-EXEC.
```

Figure 12. Example: COBOL DB2 phone application that invokes Java under z/OS Batch Runtime (Part 7 of 10)

```
  * **GET EMPLOYEES
  EXEC SQL FETCH TELE3 INTO :PPHONE END-EXEC.

  IF SQLCODE = NOT-FOUND
  * **NO EMPLOYEE FOUND
  * **PRINT ERROR MESSAGE
  MOVE '008I' TO MSGCODE
  CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
  MOVE OUTMSG TO OUTMSG2
  WRITE REPREC FROM MSG-REC2
  AFTER ADVANCING 2 LINES
  ELSE
  * **LIST SPECIFIC EMPLOYEE(S)
  PERFORM PRINT-AND-GET3
  UNTIL SQLCODE IS NOT EQUAL TO ZERO.

  * **CLOSE CURSOR
  EXEC SQL CLOSE TELE3 END-EXEC.

  PRINT-AND-GET3.
  PERFORM PRINT-A-LINE.
  EXEC SQL FETCH TELE3 INTO :PPHONE END-EXEC.
  /
  *****************************************************
  * PRINT A LINE OF INFORMATION FROM DIRECTORY *
  *****************************************************
  PRINT-A-LINE.
  * **GET INFORMATION
  MOVE LASTNAMEC TO RLNAME.
  MOVE FIRSTNAMEC TO RFNAME.
  MOVE MIDDLEINITIAL TO RMIDINIT.
  MOVE PHONENUMBER OF PPHONE TO RPHONE.
  MOVE EMPLOYEENUMBER OF PPHONE TO REMPNO.
  MOVE DEPTNUMBER TO RDEPTNO.
  MOVE DEPTNAMEC TO RDEPTNAME.
  * **PRINT INFORMATION
  WRITE REPREC FROM REPDATA
  AFTER ADVANCING 2 LINES.

  MOVE SPACES TO LASTNAMEC
  FIRSTNAMEC
  DEPTNAMEC.
  /
  *****************************************************
  * UPDATES PHONE NUMBERS FOR EMPLOYEES *
  *****************************************************
  TELEPHONE-UPDATE.
  EXEC SQL UPDATE DSN8910.VEMPLP
  SET PHONENUMBER = :NEWNO
  WHERE EMPLOYEENUMBER = :ENO END-EXEC.
  IF SQLCODE = ZERO
  * **EMPLOYEE FOUND
  * **UPDATE SUCCESSFUL
  * **PRINT CONFIRMATION
  * **MESSAGE
E INVOKE BCDTranHelper "commit"
  DISPLAY "After the BCcommit"
  Perform ErrorCheck
  MOVE '004I' TO MSGCODE
  ELSE
  * **NO EMPLOYEE FOUND
  * **UPDATE FAILED
```

Figure 12. Example: COBOL DB2 phone application that invokes Java under z/OS Batch Runtime (Part 8 of 10)

```
    * **PRINT ERROR MESSAGE
    MOVE '007E' TO MSGCODE.
    CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
    MOVE OUTMSG TO OUTMSG2.
    WRITE REPREC FROM MSG-REC2
    AFTER ADVANCING 2 LINES.
    /
    *****************************************************
    * SQL ERROR OCCURRED - GET ERROR MESSAGE *
    *****************************************************
    DBERROR.
    * **SQL ERROR
    * **PRINT ERROR MESSAGE
    MOVE '060E' TO MSGCODE
    CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
    MOVE OUTMSG TO OUTMSG1 OF MSG-REC1.
    MOVE SQLCODE TO RETCODE OF MSG-REC1.
    WRITE REPREC FROM MSG-REC1
    AFTER ADVANCING 2 LINES.
    CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
    IF RETURN-CODE = ZERO
    PERFORM ERROR-PRINT VARYING ERROR-INDEX
    FROM 1 BY 1 UNTIL ERROR-INDEX GREATER THAN 10
    ELSE

    * **MESSAGE FORMAT
    * **ROUTINE ERROR
    * **PRINT ERROR MESSAGE
    MOVE '075E' TO MSGCODE
    CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG
    MOVE OUTMSG TO OUTMSG1 OF MSG-REC1
    MOVE RETURN-CODE TO RETCODE OF MSG-REC1
    WRITE REPREC FROM MSG-REC1
    AFTER ADVANCING 2 LINES.

    ***********************************************************
    * SQL RETURN CODE HANDLING WHEN PROCESSING CANNOT PROCEED *
    ***********************************************************
    EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
    EXEC SQL WHENEVER SQLWARNING CONTINUE END-EXEC.
    EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.

F   * **PERFORM ROLLBACK
    INVOKE BCDTranHelper "rollback"
    DISPLAY "After the BCrollback"
    Perform ErrorCheck

    IF SQLCODE = ZERO

    * **ROLLBACK SUCCESSFUL
    * **PRINT CONFIRMATION
    * **MESSAGE
    MOVE '053I' TO MSGCODE
    ELSE

    * **ROLLBACK FAILED
    * **PRINT ERROR MESSAGE
    MOVE '061E' TO MSGCODE.
    CALL 'DSN8MCG' USING MAJOR MSGCODE OUTMSG.
    MOVE OUTMSG TO OUTMSG1 OF MSG-REC1.
    MOVE SQLCODE TO RETCODE OF MSG-REC1.
    WRITE REPREC FROM MSG-REC1
    AFTER ADVANCING 2 LINES.
    GO TO PROG-END.
```

Figure 12. Example: COBOL DB2 phone application that invokes Java under z/OS Batch Runtime (Part 9 of 10)

```
    ****************************************************
    * PRINT MESSAGE TEXT *
    ****************************************************
    ERROR-PRINT.
    WRITE REPREC FROM ERROR-TEXT (ERROR-INDEX)
    AFTER ADVANCING 1 LINE.

    ****************************************************
G   * Java Exception Check *
    ****************************************************
    ErrorCheck.
    Compute RETCODE = 0
    Call ExceptionOccurred
    using by value JNIEnvPtr
    returning ex
    If ex not = null then
    Call ExceptionClear using by value JNIEnvPtr
    Display "Caught an unexpected exception"
    Invoke ex "printStackTrace"
    MOVE 99 to RETURN-CODE
    End-if.
```

Figure 12. Example: COBOL DB2 phone application that invokes Java under z/OS Batch Runtime (Part 10 of 10)

Binding DB2 with Java JDBC and COBOL embedded SQL

Note: Before you begin, it is important to be familiar with the DB2 for z/OS package creation for SQLJ programs. For additional details, see the following information about writing and preparing Java programs that access DB2 for z/OS databases:

- The topic about "Programming for Java" in `http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db29.doc.java/db2z_java.htm`
- The topic about "Preparing and running JDBC and SQLJ programs" in *DB2 Application Programming Guide and Reference for Java*.
- The topic "Binding an application" in *DB2 Application Programming and SQL Guide* and in `http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db29.doc.apsg/db2z_bindplanpanel.htm`

As input, the JDBC driver of z/OS supports application package collections or a plan name. Embedded SQL in IBM Enterprise COBOL routines typically use a bound DB2 plan as input. Packages provide more flexibility by minimizing full application rebuilds when only one SQL source file is updated. Therefore, a best practice for the hybrid mixture of COBOL and Java JDBC sharing a local RRSAF attachment is to use a package list passed to the JDBC driver through the JDBC property `db2.jcc.pkList`. These JDBC properties can be passed on the Java command line using `-Dprop_name=`*`value`*. When many properties are involved, you can use the special JDBC property `-Ddb2.jcc.PropertiesFile=`*`pathname of the file`*, where the `PropertiesFile` contains the list of desired `jcc.db2.*` system properties. You can also use JDBC APIs to set properties; for more information, refer to *DB2 Application Programming Guide and Reference for Java*.

For the commands necessary to build SQLJ packages for Java programs containing SQLJ, see “Commands for SQLJ program preparation” on page 44.

There is considerable flexibility when binding with existing packages and DBRM members, or both. To bind a COBOL program containing embedded SQL, which has been preprocessed or co-processed to produce DBRM member XMPCOBJX (for instance, COBOL extended with Java JDBC), you can use --.

```
//BINDCOBX JOB (1),'name'
//             NOTIFY=&SYSUID,
//             MSGCLASS=X,
//             CLASS=A,
//             REGION=0M,
//             TIME=120
//BINDEXE  EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//DBRMLIB  DD DSN=SUIMGJB.DBRMLIB.DATA,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//REPORT   DD SYSOUT=*
//SYSIN    DD *
//SYSTSIN DD *
 DSN SYSTEM(DSN9)
  BIND PACKAGE(XMPCOBJX) MEMBER(XMPCOBJX) -
  ACT(REP) ISO(CS) CURRENTDATA(YES) ENCODING(EBCDIC)
/*
```

Figure 13. Example: JDBC-only case

Using the example in Figure 13, you now have a new COBOL collection named XMPCOBJX.*. This can be passed to z/OS Batch Runtime as the system property `db2.jcc.pkList`, which can be appended to the default JDBC-provided `NULLID` collection. On the Java command line, for example, this would be seen as follows:

```
-Ddb2.jcc.pkList=NULLID.*,XMPCOBJX.*
```

You should also grant package privileges, according to the specific security standards that are in place at your installation. Using the example in Figure 13, you would specify the following statement, where *authid* can be either a user ID or secondary ID, such as a RACF (SAF) group name.

```
GRANT EXECUTE ON PACKAGE XMPCOBJX.* TO authid
```

Commands for SQLJ program preparation

To build SQLJ packages for Java programs that contain SQLJ embedded SQL, knowledge and use of the following commands is a must:

sqlj - SQLJ translator
The `sqlj` command translates an SQLJ source file into a Java source file and zero or more SQLJ serialized profiles. By default, the `sqlj` command also compiles the Java source file.

db2sqljcustomize - SQLJ profile customizer
The `db2sqljcustomize` command augments the profile with DB2-specific information for use at run time. It processes an SQLJ profile, which contains embedded SQL statements. By default, `db2sqljcustomize` produces four DB2 packages: one for each isolation level.

Remember, also, to include `SQLJ.JAR` in the classpath set up of your BCDBATCH JCL.

For the complete details, syntax, authorization, and parameters, see the topic on "JDBC and SQLJ reference information" in *DB2 Application Programming Guide and Reference for Java*.

Chapter 6. Troubleshooting for z/OS Batch Runtime

In addition to the standard z/OS messages (in the format BCD*nnnnx*, where *nnnn* is the message number and *x* is the message severity), z/OS Batch Runtime provides logging and tracing facilities for troubleshooting problems. The following topics explore trace and logging in more detail. For more information about messages, see *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

Trace facilities for z/OS Batch Runtime

All z/OS Batch Runtime classes are designed to use the standard Java trace facilities available in the `java.util.logging` package. At a minimum, z/OS Batch Runtime traces entry and exit to all significant methods, all exceptions, and all significant events. Tracing is controlled through a system property or by the logging configuration file, which by default is specified in the `jre/lib/logging.properties` file. You can override the location of the file using the following Java system property when you invoke z/OS Batch Runtime:

```
java.util.logging.config.file
```

Use a trace to diagnose problems in z/OS Batch Runtime. Obtain the trace using the following system property:

```
com.ibm.zos.batch.container.BCDTraceConfig=trace-level
```

The property values for *trace-level* are ALL, which indicates that all events will be traced, or NONE, which indicates no tracing. When diagnosing problems, use a trace level of ALL.

Log facilities for z/OS Batch Runtime

z/OS Batch Runtime provides a verbose mode to provide additional messages that can assist in diagnosing batch runtime problems. When running in verbose mode, all messages are created for all commit and rollback requests. Messages are written to `//BCDOUT`.

Signalling and exception handling by z/OS Batch Runtime

COBOL applications have a specific signal or error condition handling process. Java has a defined signal handling process as well as a set of JNI processes for signal and error condition handling. Language Environment also has application programming interfaces (APIs) for application code that allows you to customize condition handling to override the default settings.

z/OS Batch Runtime uses the JVM startup option -XCEEHDLR. This option informs the JVM to register a stack-based Language Environment condition handler before COBOL JNI calls. It is then able to translate potentially-recoverable Language Environment exceptions into a Java exception and pass it back to the calling Java code. z/OS Batch Runtime catches and reports percolated runtime exceptions out of the Java application.

Appendix. Accessibility

Publications for this product are offered in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when using PDF files, you may view the information through the z/OS Internet Library website or the z/OS Information Center. If you continue to experience problems, send an email to mhvrcfs@us.ibm.com or write to:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
U.S.A.

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer or Library Server versions of z/OS books in the Internet library at:

`http://www.ibm.com/systems/z/os/zos/bkserv/`

Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

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 give 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 character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 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:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

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 information 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.

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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

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 illustrates 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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS™, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Index

Product Number: 5694-A01

Printed in USA

SA23-2270-01