



Introduction

In the z/OS environments of today, installations have a need to re-engineer existing native z/OS COBOL applications to incorporate the Java language in order to take advantage of its larger developer skill base and its many language features. Moreover, a mixed z/OS COBOL and Java batch application usually share a DB2 connection and use relevant language APIs to communicate with DB2 in the same unit of work (UOW). Additionally, during the running of such a re-engineered program, Embedded Structured Query Language (SQL) DB2 access in Enterprise COBOL and Java Database Connectivity (JDBC), Structured Query Language for Java (SQLJ) DB2 access in Java, or both must coexist transparently.

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.

The Batch Runtime initializes the environment for COBOL and Java interoperability as follows:

- Sets up the job step under a Resource Recovery Services (RRS) managed global transaction (see: #57 zTidBits - Resource Recovery Services)
- Calls the primary COBOL or Java application after you have initialization of the 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 (5655-R31)
- Enterprise COBOL Version 4.2 and above

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

Planning for z/OS Batch Runtime

When planning use of z/OS Batch Runtime, a good application to consider is a native z/OS COBOL application that you want to enhance or replace function with Java method calls. The entire application code must be single threaded.

You invoke z/OS Batch Runtime through JCL that invokes the job BCDBATCH. BCDBATCH, in turn, invokes the JZOS launcher to initialize the Java environment, (see: #34 zTidBits Java Batch).

Because BCDBATCH invokes JZOS, one level of the JZOS launcher exists for each Java SDK level and bit mode. You can define the level of the JZOS launcher with a symbolic, so your installation can update the symbolic as new levels of the Java SDK are added or made the default.

Configuring Java

You 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 configure the z/OS Batch Runtime using JCL and the environment variables it provides. To set the Java runtime variables in the environment script, use the IBM_JAVA_OPTIONS environment variable.

Additionally, JZOS defines several environment variables that allow you to control the encoding of output written to stdout and stderr, Java options that JZOS uses when it creates the JVM, and main method program arguments.

NOTE: 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



Improving Java start up time

For short-running jobs, improving Java start up time is important. This is especially true when running hundreds or thousands of small Java batch jobs, the Java start up elapsed time and CPU time become an important performance measurement.

Using the following Java options can make it possible to reduce the Java startup times for applications that start a new JVM frequently:

- Xquickstart Java option
- Shared classes and AOT Java options

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

Java environment variables for z/OS Batch Runtime

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

• CLASSPATH

The application must set the CLASSPATH to include the Batch Runtime jar files using the CLASSPATH environment variable specified in the BCDBATCH JCL procedure. The configuration script automatically configures the CLASSPATH based on the exported BCD_HOME variable in the BCDBATCH JCL procedure.

• LIBPATH

The application must set LIBPATH to the location of the DLLs for z/OS Batch Runtime in the JCL procedure. The configuration script performs the function.

NOTE: For this release, z/OS V1R13, z/OS Batch Runtime supports *only* 31-bit applications; you must use the 31-bit JVM.

Quick guide to BCDBATCH

The following quickstart table summarizes the main JCL statements for the BCDBATCH job:

Job Control Language	Explanation
//BCDBATCH JOB (1), "name"	The JCL that invokes the job BCDBATCH
//STEPLIB DD DSN=hlq.yourapp.loadlib,DISP=SHR // DD DSN=hlq.jzos.loadlib,DISP=SHR	Add any load libraries your application requires, such as the data set containing your COBOL application load modules to the STEPLIB. If the JZOS Java launcher is not installed in the z/OS linklst, add a STEPLIB for it.
//BATCH EXEC BCDPROC,REGION=0M	BCDPROC is the batch container JCL procedure.
//STDENV DD *	Specifies the environment variables used for this run including CLASSPATH and LIBPATH.
//BCDIN DD *	Specifies a file containing the batch configuration options. Remember that some support elements obtain their options from Java system properties. You can control z/OS Batch Runtime through the options that you specify on the //BCDIN JCL statement.

JCL summary for BCDBATCH job

Note: A current sample of BCDBATCH job for z/OS Batch Runtime, is always in SYS1.SAMPLIB in the latest z/OS release (as of this writing V1R13).



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

Before z/OS Batch Runtime processes any COBOL or Java applications during start up, it invokes the initialization function that is supported for the database resource.

During this processing, the Java Database Connectivity (JDBC) driver for DB2 establishes the connection with a single resource attachment 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* Batch Runtime attachment to access the DB2 resources. For COBOL applications, DB2 ensures this common connection through either pre-processor directives or through run time processing.

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 transaction synchronization:

- *beforeCompletion*. This method is invoked before the transaction process starts.
- *afterCompletion*. This method is 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

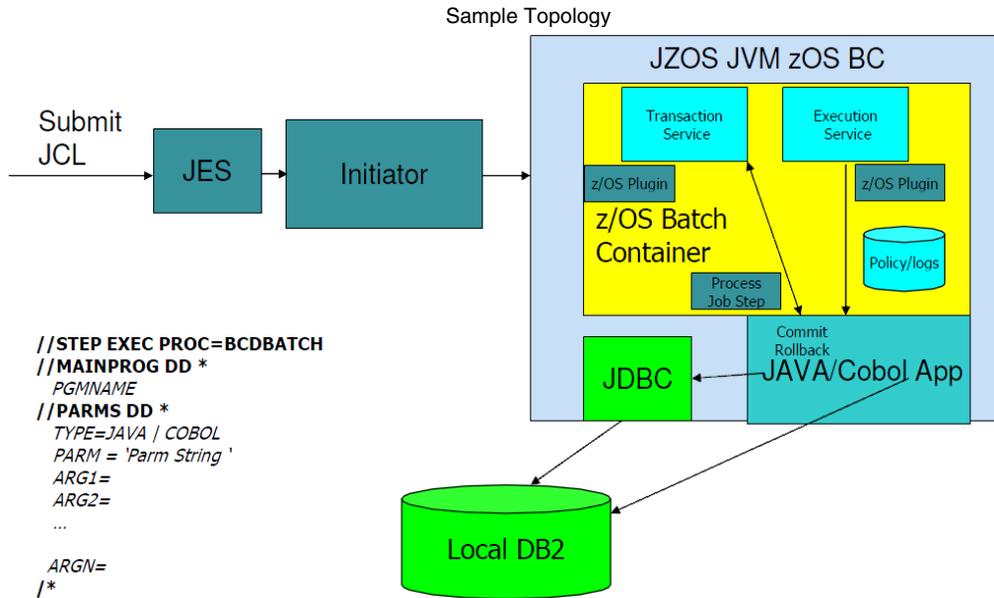
COBOL invokes Batch Runtime methods for commit and rollback. For COBOL applications, z/OS Batch Runtime offers callable procedures for commit and rollback of a transaction. Before committing the unit of work, z/OS Batch Runtime invokes the *beforeCompletion* method on the JDBC to indicate the start of 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.



Uses same JCL you have known

Summary

- Ability to replace/add functions in current 3GL DB2 (e.g. COBOL DB2) application inventory with new Java DB2 code
 - Requires local attach z/OS DB2 connection sharing for common DB2 access
 - Requires UOW (Transactional) integrity among the application components
- A generalized solution without requiring a specific run-time or middleware, i.e. a pure batch environment
- Implementation requires little or no changes to existing code!
- Only requires special callbacks for commit/rollback

— — —