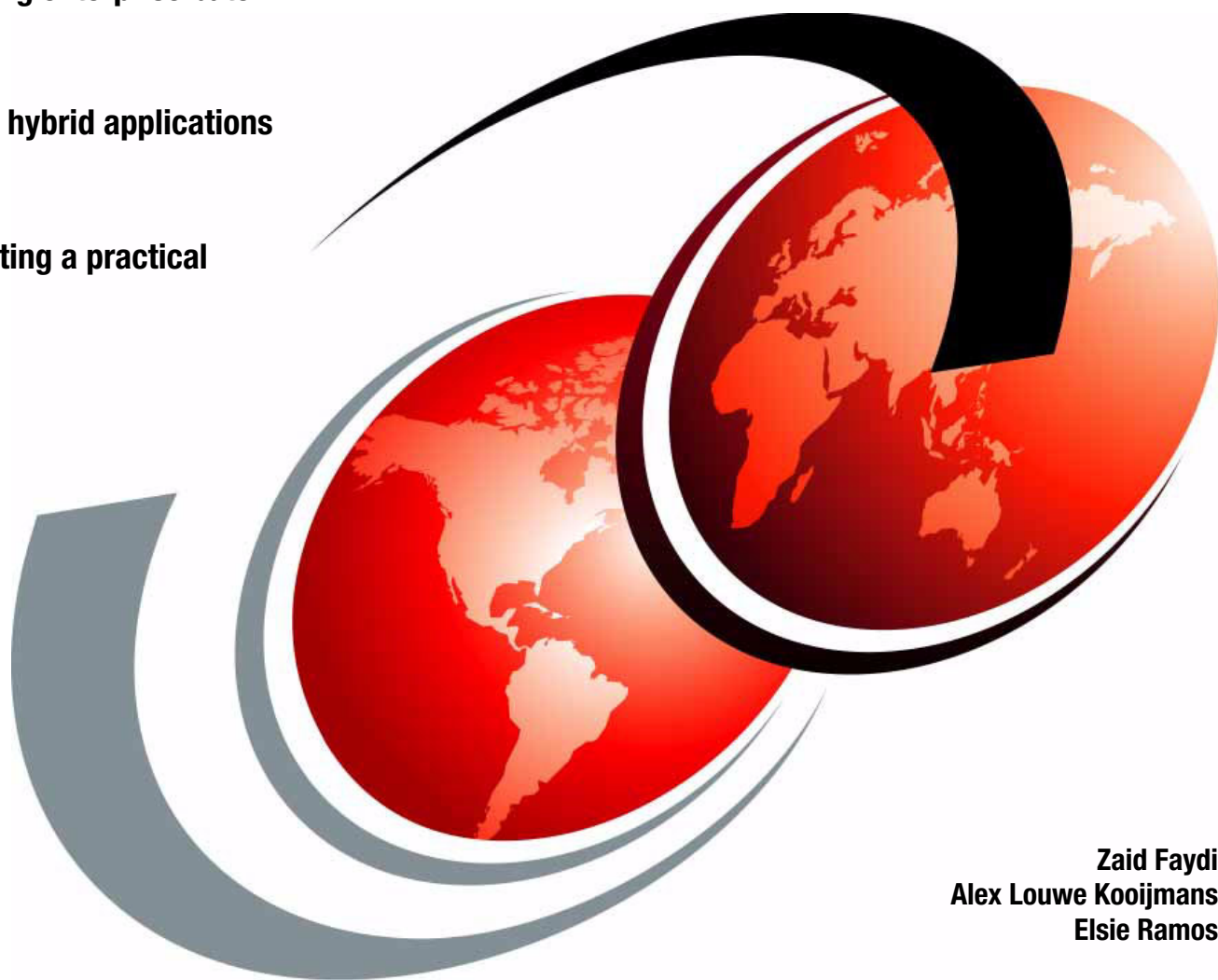


New Ways of Running IBM z/OS Batch Applications

Modernizing enterprise batch

Designing hybrid applications

Implementing a practical
example



Zaid Faydi
Alex Louwe Kooijmans
Elsie Ramos

Redbooks



International Technical Support Organization

New Ways of Running IBM z/OS Batch Applications

May 2013

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

First Edition (May 2013)

This edition applies to the following software levels:

- ▶ z/OS Version 1 Release 12 and Release 13
- ▶ IBM 64-bit SDK for z/OS, Java Technology Edition, V6
- ▶ Rational Application Developer V8.0.3 iFix1

© Copyright International Business Machines Corporation 2013. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	v
Trademarks	vi
Preface	vii
Authors	vii
Now you can become a published author, too!	viii
Comments welcome	viii
Stay connected to IBM Redbooks	viii
Chapter 1. Modernizing enterprise batch with hybrid Java/COBOL applications	1
1.1 What do we mean by a hybrid batch application	2
1.2 Procedural language or Java	2
1.2.1 Benefits of a procedural language	2
1.2.2 Benefits of Java	2
1.3 Why hybrid batch applications?	3
1.3.1 Skills	3
1.3.2 Reuse of code	3
1.3.3 Access to more functionality	3
1.3.4 Exploiting speciality engines	3
1.4 Transition considerations	3
1.4.1 Governance considerations	4
1.5 Introduction to z/OS batch runtime	4
Chapter 2. What is z/OS Batch Runtime?	5
2.1 Topology	6
2.2 Usage and invocation	6
2.2.1 Invoking the primary application	6
2.2.2 Passing arguments	7
2.2.3 Viewing application and container output	8
2.2.4 Handling return codes and Java exceptions	9
2.2.5 Commit rollback services	10
2.2.6 Terminating managed applications	10
2.2.7 Sharing a DB2 connection	10
2.2.8 Binding COBOL	11
2.3 Required software	11
2.4 Invoking z/OS Batch Runtime	11
2.5 Restrictions	13
2.6 Design considerations	13
2.6.1 Dynamic SQL versus static SQL	13
2.6.2 Data recovery	15
2.6.3 JDBC Type 2	15
2.7 Migrating a hybrid COBOL/Java application	15
Chapter 3. End-to-end development scenario	17
3.1 Software used in the scenario	18
3.2 Infrastructure setup	18
3.2.1 z/OS Batch Runtime verification	18
3.2.2 Java	20
3.3 Setting up the development environment	20

3.3.1	Connecting to z/OS from Rational Developer for System z	21
3.3.2	Creating the HFS directories.	24
3.3.3	Creating the z/OS project and MVS and UNIX subprojects	25
3.3.4	Creating the DB2 QUERIES table.	30
3.3.5	Creating sample data sets	37
3.4	Developing COBOL, Java, and JCL	41
3.4.1	Adding COBLOAN	41
3.4.2	Generating the JCL to compile, link, bind, and run COBLOAN.	44
3.4.3	Creating the Java launcher and copying the JNI copybook	48
3.4.4	Creating the PdfCreator class.	49
3.4.5	Compiling PdfCreator	51
3.4.6	Running COBLOAN in the container.	53
3.4.7	Solution to reflect an exception in the return code	59
	Appendix A. Additional material	63
	Locating the Web material	63
	Using the Web material	63
	System requirements for downloading the Web material	63
	Downloading and extracting the Web material	64
	Related publications	65
	IBM Redbooks publications	65
	Other publications	65
	Online resources	65
	Help from IBM	65

Notices

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

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites 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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:


This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CICS®
DB2®
IBM®
MVS™

Rational®
Redbooks®
Redbooks (logo) ®
System z®

WebSphere®
z/OS®

The following terms are trademarks of other companies:

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

Mainframe computers play a central role in the daily operations of many of the worlds largest corporations. Batch processing is still a fundamental, mission-critical component of the workloads that run on the mainframe and a large portion of the workload on IBM® z/OS® systems is processed in batch mode.

This IBM Redbooks® publication is the second volume in a series of four in which we describe new technologies introduced by IBM to facilitate the use of hybrid batch applications that combine the best aspects of Java and procedural programming languages such as COBOL. This volume specifically focuses on z/OS batch runtime.

The audience for this book includes IT architects and application developers, with a focus on batch processing on the z/OS platform.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Zaid Faydi is a Software Engineer who is currently Test Manager at CSC in Perth, Australia. Previously he worked at IBM for 6 years. His areas of expertise include z/OS application quality assurance and Java programming.

Alex Louwe Kooijmans is a Senior Architect at the Financial Services Center of Excellence at IBM Systems and Technology Group. Prior to this position, he spent almost 10 years in the International Technical Support Organization leading IBM Redbooks projects, teaching workshops, and running technical events with a focus on using the IBM mainframe in new ways. Alex also worked as Client Technical Advisor to various banks in the Netherlands and performed several job roles in application development. His current focus is on modernizing core banking systems and the role of IBM's current mainframe technology.

Elsie Ramos is a Project Leader at the International Technical Support Organization, Poughkeepsie Center. She has over 30 years of experience in IT, supporting various platforms, including IBM System z® servers.

Thanks to the following people for their contributions to this project:

Rich Conway, Michael Schwartz
International Technical Support Organization, Poughkeepsie Center

Gary Puchkoff
Software Architect, IBM Systems and Technology Group. z/OS New Technology
IBM Poughkeepsie, USA

Stephen Henkels, Ken Jonas, Ruth Ray
IBM Poughkeepsie, USA

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:
ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



Modernizing enterprise batch with hybrid Java/COBOL applications

Mainframe computers play a central role in the daily operations of many of the world's largest corporations. Although other forms of computing are used extensively in various business capacities, the mainframe occupies a coveted place in today's e-business environment. In banking, finance, healthcare, insurance, public utilities, government, and a multitude of other public and private enterprises, the mainframe computer continues to form the foundation of modern business.

Batch processing is still a fundamental, mission-critical component for most companies in every industry. The IBM Redbooks publication, *Batch Modernization on z/OS*, SG24-7779, describes aspects of modern batch processing and why it might be necessary to change the current batch process to achieve modern business requirements. In this book, we focus specifically on technology introduced by IBM to facilitate the use of hybrid batch applications that combines the best aspects of Java and procedural programming languages such as COBOL.

In this chapter, we introduce the concept of hybrid batch applications and discuss the benefits of using Java in combination with COBOL. We also introduce the z/OS Batch Runtime, a new z/OS V1.13 base component, which facilitates development and deployment of hybrid Java/COBOL applications without the requirement of having IBM WebSphere®, IBM CICS® or any other container. This provides hybrid COBOL/Java batch applications with the capability to share an IBM DB2® connection and execute transactions as a single unit of work (UOW).

1.1 What do we mean by a hybrid batch application

In this book, the term *hybrid batch application* is used for an application that combines programs written in a procedural language, such as COBOL, with programs written in Java. The reasons for choosing this approach for application development vary depending on the starting point for the development of the application:

- ▶ If the starting point is a traditional z/OS batch application that is written in COBOL, then the reason to make some calls into Java might be to take advantage of the many available off the shelf software solutions that perform tasks such as XML conversion, PDF generation, email, and so on.
- ▶ If the starting point is the development of a modern Java batch application, then there could be reasons to still write new or reuse certain existing routines in COBOL. These reasons might be performance-related issues, or the need to access native resources on z/OS, such as accessing a VSAM data set.

1.2 Procedural language or Java

In this section, we highlight the main reasons to choose a particular programming language, and why it might be appropriate to use both a procedural language and Java in the same batch application.

1.2.1 Benefits of a procedural language

A significant number of current business applications on the mainframe are written using traditional languages such as Assembler, COBOL, and PL/I. The data serving capabilities of System z provide a reliable foundation for an optimized IT infrastructure as required in today's business environments. COBOL with its procedural programming model provides a huge strength for processing massive amounts of data. Programming languages such as COBOL provide exactly the functionality that is needed, and in a very efficient way, for massive database and file reads and updates, and this is still what many batch programs are all about. So, if your primary concern is to be able to perform massive database and file access in the fastest and most reliable way possible, then procedural programming languages such as COBOL are the best fit.

1.2.2 Benefits of Java

One of the major benefits of Java is its current popularity within the IT industry. This popularity means that there are plentiful skills and resources in the marketplace, which helps to keep development costs down. Third-party vendors produce software packages that can be bought off the shelf and there is also good tooling support. Some functions that are considered challenging to do in a language like COBOL, such as generating PDF documents or sending emails, are easy to do using readily available open source Java packages. Java programs running on z/OS are eligible for offload to a System z Application Assist Processor (zAAP), which can help to reduce the monetary cost of running a workload on z/OS.

Refer to *Batch Modernization on z/OS*, SG24-7779 for a broad discussion about the benefits of using Java in an existing z/OS environment.

1.3 Why hybrid batch applications?

There are various reasons why it makes good business sense to consider hybrid batch applications. One reason is the need to modernize batch processing while continuing to realize the benefits of the existing, stable, efficient programs that you already have running within your organization. Some other reasons are the availability of skills and access to more functionality.

1.3.1 Skills

In the z/OS environments of today, many organizations want to re-engineer existing native z/OS COBOL applications to incorporate Java in order to benefit from its larger developer skill base and its many language features. Companies want to take advantage of the existing Java skills to create new batch applications because they can easily find the people and resources needed to make those applications a success. At the same time, it does not make business sense to rewrite existing COBOL applications. Therefore, there is the need to integrate Java and procedural languages such as COBOL in a seamless manner.

1.3.2 Reuse of code

Reuse of code can mean the reuse of existing assets built up over time in your organization, or utilizing a language such as Java to reuse functionality developed by third-party vendors or open source software packages.

1.3.3 Access to more functionality

Java packages are available to parse XML, invoke web services, build PDF documents, send emails, and perform many other functions. As organizations attempt to modernize and to provide more of these types of functions to their customers, it makes sense to enhance existing procedural batch applications by exploiting the thriving Java community and thus making the introduction of these technologies as easy as possible.

Procedural languages such as COBOL are good at manipulating data, reading and writing records from z/OS resources such as VSAM files and relational databases. It might make sense to call out from a modern Java batch application to a COBOL routine to manipulate data and access the z/OS resources that are needed for the batch job.

1.3.4 Exploiting speciality engines

CPU-intensive routines can be written in Java to enable them to offload to zAAP processors. zAAP processors have a one time charge pricing model; therefore, by offloading processing to Java and then to a zAAP, the overall MIPS usage of the z/OS machine can be lowered, thus lowering the cost of running a workload.

1.4 Transition considerations

When moving to any new technology, a staged approach is recommended, to ensure that any unforeseen challenges have minimal impact on the business. For batch modernization using hybrid batch applications, consider starting with an application that is not on the critical path for the timeline of batch jobs that must complete within a batch window.

1.4.1 Governance considerations

This chapter has highlighted some of the reasons it makes sense to build hybrid applications. At the same time, the addition of hybrid applications is likely to introduce additional governance challenges that must be considered, such as the requirement to have both Java and COBOL skills available for application maintenance and problem determination, and build and test processes and tools that encompass both programming languages.

1.5 Introduction to z/OS batch runtime

Batch processing on mainframe computers continues to be the cornerstone of many of the world's largest corporations. Government agencies, the military, and commercial enterprises use COBOL to run their business. Updating COBOL applications today to satisfy new business requirements can be difficult because of the dwindling number of COBOL developers and the limited support by the COBOL compilers and libraries for modern technologies. Fortunately, the IBM Enterprise COBOL for z/OS compiler features COBOL interoperability with Java, allowing developers to replace and extend the functionality of COBOL using Java's rich features and a global developer population.

When the benefits of two or more programming languages are combined, the result is called a *hybrid application*. This extended functionality can increase the complexity and present some domain-specific pitfalls. One such pitfall is the loss of transactional integrity when a unit of work spans different components. For example, a COBOL application issues an UPDATE change to a DB2 table, then invokes Java code that will issue another required UPDATE to the same resource. If the UPDATE in the Java code fails, the COBOL UPDATE will not be rolled back because the two transactions were not defined as a single unit of work.

Natively, executing as a single transaction across more than one database connection increases the complexity of your code because transaction checking and failure recovery logic must be added.

To extend a batch database COBOL application with Java functionality and also ensure data integrity, consider using the z/OS Batch Runtime. In a nutshell, this container offers a managed environment in which to execute a hybrid Java/COBOL program as a single unit of work. A single connection to a local database is established at startup and is shared across both the Java and COBOL portions of the program. Commit and rollback services are available to your program to either discard your changes or make them permanent.

This book provides a detailed discussion of this container. It also covers design considerations and provides a development scenario to illustrate how this container can be used to extend a batch COBOL application with Java functionality using the modern development tooling of IBM Rational® Developer for System z.



What is z/OS Batch Runtime?

In this chapter, we discuss the Batch Runtime environment, a new component that was introduced in z/OS V1R13. This component provides a new option for running batch work that is focused on Java. This new platform-neutral programming model, called the *z/OS Batch Runtime*, is a managed environment for Java that allows interoperability with COBOL. Batch code written to this new standard runs natively on z/OS. Executing DB2 hybrid applications in z/OS Batch Runtime simplifies the code, helps ensure the integrity of your data, and overcomes some common batch pitfalls.

The z/OS Batch Runtime initializes the environment for COBOL and Java interoperability by doing the following:

- ▶ Setting up the job step under a Resource Recovery Services (RRS) managed global transaction
- ▶ Calling the primary COBOL or Java application after initialization of the environment.

Refer to *Batch Modernization on z/OS*, SG24-7779, for further details about z/OS Batch Runtime and the updates in z/OS 1.13 to improve BATCH job processing.

2.1 Topology

The topology of the z/OS Batch Runtime is displayed in Figure 2-1. It depicts the following job flow:

- ▶ Submission of the JCL
- ▶ Execution of the BCDBATCH Proc (Java batch application that invokes z/OS Batch Runtime)
- ▶ Instantiation of the z/OS batch container
- ▶ Use of RRS to local DB2

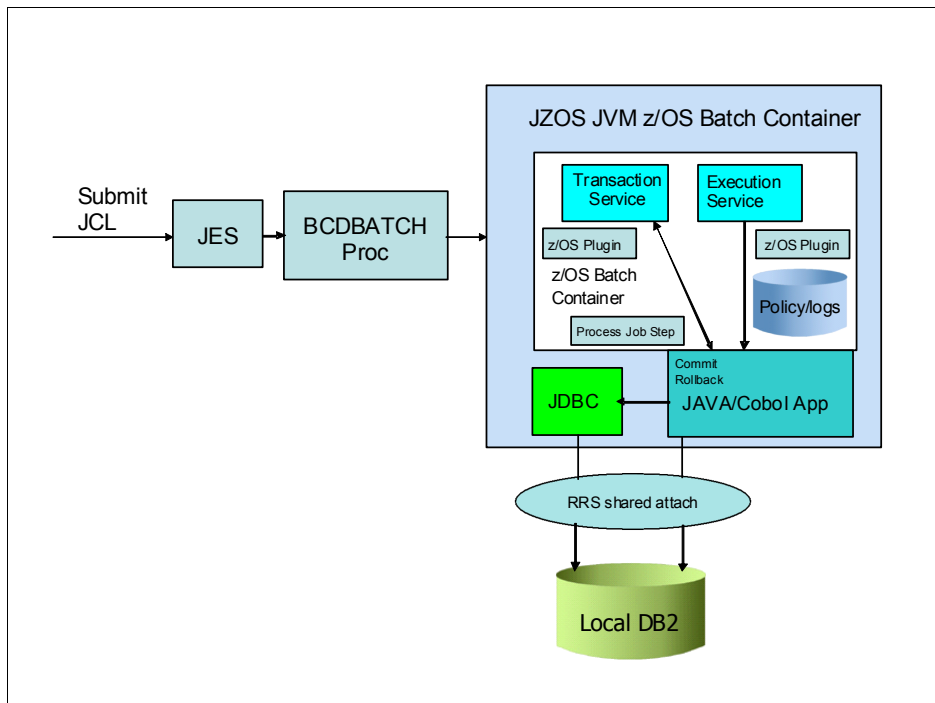


Figure 2-1 Topology of z/OS Batch Runtime

The z/OS Batch Runtime provides an environment where your DB2 COBOL and Java applications can share a single database connection. The z/OS Batch Runtime also provides commit/rollback services by leveraging the z/OS Resource Recovery Services (RRS) and the Java Database Connectivity (JDBC) driver.

2.2 Usage and invocation

The following sections provide detailed information about how the z/OS Batch Runtime works.

2.2.1 Invoking the primary application

The primary application refers to the first or main application that executes in the batch container. The Batch Container is implemented in Java and it installs into the existing path `/usr/lpp/bcp`. In the z/OS Batch Runtime, a Java batch application `BCDBATCH` invokes the

JZOS launcher to initialize a JVM before calling the primary application. The JZOS launcher is a tool that runs Java applications directly as batch jobs.

In your JCL, you must specify the Java archive (JAR) and dynamic link library (DLL) files necessary to run your primary application and the z/OS Batch Runtime.

The z/OS Batch Runtime configuration options are defined in the BCDIN data definition name (DDNAME). To specify the primary application, set the `bcd.applicationName` option to the name of your application and specify the programming language in `bcd.applicationLanguage` (Refer to Example 2-1).

Note: Only COBOL and Java are currently supported in this release.

2.2.2 Passing arguments

In this section, we demonstrate how to pass arguments to either COBOL or Java applications.

- ▶ To pass a parameter to your primary COBOL application (up to 100 characters) declare and set the `bcd.applicationArgs.1` option as shown in Example 2-1.

Example 2-1 z/OS Batch Runtime configuration for a COBOL application

```
bcd.applicationLanguage=COBOL
bcd.applicationName=RECMANGR
bcd.applicationArgs.1=Zaid Faydi
```

- ▶ To pass a string array to your main Java application method, use the same option and increment the index for each element, as shown in Example 2-2.

Example 2-2 z/OS Batch Runtime configuration for a Java application

```
bcd.applicationLanguage=JAVA
bcd.applicationName=RecordManager
bcd.applicationArgs.1=Zaid
bcd.applicationArgs.2=Faydi
```

RecordManager class

The RecordManager class contains the simple main method shown in Example 2-3.

Example 2-3 Record Manager main method

```
public static void main (String args[]) {
    System.out.println("Firstname = >" + args[0] + "<");
    System.out.println("Surname   = >" + args[1] + "<");
}
```

Executing the RecordManager produces the output shown in Example 2-4.

Example 2-4 RecordManager execution output

```
Firstname = >Zaid<
Surname   = >Faydi<
```

Exceptions

If you reference an index that does not exist in the array, an exception occurs which terminates the container with return code 12. For further details, refer to “Handling return codes and Java exceptions” on page 9.

If the `bcd.applicationArgs.2` option is omitted, an `ArrayIndexOutOfBoundsException` occurs as shown in Example 2-5. This happens because a single element String array is passed to the `RecordClass` main method when it references the first and second elements.

Example 2-5 Exception occurs due to missing option

```
BCD0102E Exception occurred: java.lang.reflect.InvocationTargetException
...
Caused by: java.lang.ArrayIndexOutOfBoundsException: Array index out of range: 1
  at RecordManager.main(RecordManager.java:4)
```

If the `bcd.applicationArgs.2` option is declared without a value, as illustrated in Example 2-6, then an empty string is passed.

Example 2-6 applicationArgs without a value

```
bcd.applicationLanguage=JAVA
bcd.applicationName=RecordManager
bcd.applicationArgs.1=Zaid
bcd.applicationArgs.2=
```

An empty string produces the output shown in Example 2-7.

Example 2-7 Output when an empty string is passed

```
Firstname = >Zaid<
Surname   = ><
```

Note: The `bcd.applicationArgs` option can be omitted from the JCL without affecting the application or the run time if no references are made to the application arguments.

2.2.3 Viewing application and container output

The following is a list of the z/OS Batch Runtime JES output with a description of what each data definition (DDNAME) shows:

SYSOUT	Java Virtual Machine (JVM) output and COBOL DISPLAY output
STDOUT	Java console output (System.out) from applications
STDERR	Java error output (System.err) from both application and container
BCDOUT	z/OS Batch Runtime output
BCDTRACE	z/OS Batch Runtime trace output

2.2.4 Handling return codes and Java exceptions

The z/OS Batch Runtime can terminate with one of the following codes:

- 00** Denotes successful processing.
- 08** Primary application returns a code greater than 0.
- 12** Could not launch the primary application or an unhandled abend or exception has occurred.

Return code and exception handling by the primary application is important to ensure an accurate representation of execution. Both need to be programmatically implemented and taken into consideration when designing hybrid applications. The primary application decides the final return code to send based on its own processing and the values returned from the applications it called.

Generally, the highest return code is cascaded to the primary application and returned. For example, a COBOL application that calls a Java method that returns an integer value needs to move this value to the return-code register if it is higher than the current return-code value. If the Java return value is a boolean and denotes the success of execution, then one possible implementation is to set the COBOL return code register to 12 only if the value is *false*.

Failure to handle return values from other components returns the value of the return-code register upon completion of a COBOL primary application, which in some cases might not reflect the overall execution of the application.

If the called Java method has an exception during processing that the COBOL primary application does not handle, then the container terminates with a return code of 12. Not all exceptions warrant a return code of 12; therefore, handling them appropriately is essential. It requires the primary application to decide whether an exception should cause the container to return a 0 or an 8. Despite that an exception occurred, the action was successful, so a return code of 0 can be returned, as well as a log message to warn about the cause of the exception.

Example 2-8 provides a simple code snippet of a COBOL application that handles the return value of a Java method and checks to see if an exception has occurred.

Example 2-8 Sample code to handle return code and exceptions from Java

```
* Calling the returnRC Java method
  DISPLAY 'COBOL: Calling returnRC with parameter' INPUT-VALUE
  INVOKE DemoJava "returnRC"
  USING BY VALUE Parameter
  RETURNING RC
  IF RC > RETURN-CODE THEN
    MOVE RC TO RETURN-CODE.
  PERFORM EXCEPTION-CHECK.
  GOBACK.
MAIN-FINISH.
* Checks if an Exception has occurred
EXCEPTION-CHECK.
  CALL ExceptionOccurred
  USING BY VALUE JNIEnvPtr
  RETURNING JavaException
  IF JavaException NOT = NULL THEN
    CALL ExceptionClear
    USING BY VALUE JNIEnvPtr
```

```
        DISPLAY "An Exception was caught from Java"
    ELSE
        DISPLAY "No Exception caught".
GOBACK.
```

2.2.5 Commit rollback services

The z/OS Batch Runtime uses the two-phase commit protocol to coordinate whether to commit or roll back (abort) a transaction.

Commit	A command that makes changes to a database permanent
Transaction	A collection of changes between successive commit commands
Rollback	A command that cancels the collection of changes since the last commit command call (transaction)

Rather than implementing your own two-phase commit protocol in your hybrid applications, you can exploit the resource managers provided by the z/OS Resource Recovery Services (RRS). RRS is a general global sync point manager, and a component of z/OS that allows applications to guarantee that data changes have been made or backed out of a resource, such as a DB2 database.

The Java Database Connection (JDBC) is a resource manager that exploits RRS to read and change database data, and take actions such as committing or backing out changes.

The z/OS Batch Runtime implements RRS and JDBC calls to provide your applications with helper methods to simplify the usage of commit rollback. The commit and rollback methods reside in the `com.ibm.batch.spi.UserControlledTransactionHelper`. The use of these helper methods is illustrated in “End-to-end development scenario” on page 17.

2.2.6 Terminating managed applications

When the primary application is invoked, it runs on the same thread/task control blocks (TCB) as the z/OS Batch Runtime. Subsequent calls to other applications also run on the same thread/TCB. The Java `System.exit` method and COBOL **STOP RUN** must therefore not be used in your hybrid application because it will kill the thread/TCB and terminate the batch container.

2.2.7 Sharing a DB2 connection

A special mode of the JDBC type 2 (local) connectivity in conjunction with RRS is used to identify z/OS Batch Runtime transactions. When a database connection request is made from an application, it is detected and runs via the database attachment created at startup. A single DB2 attachment is shared when a single database connection is established by the batch container at startup and is used by applications when they communicate with the database. Only one single DB2 resource attachment is created and used by all the applications in the run time; therefore, only one local DB2 database is supported.

Note: Only local DB2 databases are supported in this release because applications and DB2 resources must exist on the same z/OS image.

A new thread or TCB created that executes a database transaction is not identified as a z/OS Batch Runtime transaction. A new attachment is created and any transaction executed is out

of scope of the global commit/rollback services that the container provides (via RRS and JDBC). We therefore discourage executing multi-threaded database applications in this container.

2.2.8 Binding COBOL

z/OS Batch Runtime binds COBOL applications to a default JDBC collection of packages called NULLID. You can include your own packages by using the IBM Data Server Driver for JDBC `Ddb2.jcc.pkList` configuration. Example 2-9 demonstrates how to concatenate the package BANK to the package list in your run JCL.

Example 2-9 Concatenate package in JCL

```
IJO="$IJO -Ddb2.jcc.ssid=DB0V -Ddb2.jcc.pkList=BANK.*,NULLID.*"
```

Cleanup processing

If the application terminates with an unhandled exception or an abend, then the z/OS Batch Runtime rolls back all the outstanding database changes. Otherwise, it commits them regardless of the return code.

2.3 Required software

The following software is required to run z/OS Batch Runtime:

- ▶ IBM 31-bit SDK for z/OS, Java Technology Edition, V6.0.1(5655-R31)
- ▶ Enterprise COBOL version 4.2 and later
- ▶ One of the following versions and levels of DB2:
 - 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

2.4 Invoking z/OS Batch Runtime

z/OS Batch Runtime is invoked through a batch JCL. *BCDPROC* is the batch container JCL procedure. A sample JCL BCDPROC is shown in Example 2-10.

Example 2-10 BCDPROC procedure

```
//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 = *
/** *
/** $LO=BATCH,HBB7780,100324,KDKJ: *
/** *
/** *
/*******
/**JAVA EXEC PGM=JVMLDM&VERSION,REGION=OM,
/** 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=*
/**

```

The JCL that invokes z/OS Batch Runtime invokes the job *BCDBATCH*. *BCDBATCH*, in turn, invokes the JZOS launcher to initialize the Java environment.

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

Batch runtime BCDBATCH can be used to launch a user COBOL or Java application, which itself can call another COBOL or Java application, and so on. The //BCDIN DD * JCL specifies the file containing the batch configuration options.

Sample BCDBATCH and BCDIN JCLs with explanations are available in Chapter 2 of *z/OS Batch Runtime: Planning and User's Guide*, SA23-7270.

Note: A current sample of the BCDBATCH job for z/OS Batch Runtime is always available in SYS1.SAMPLIB in the latest z/OS release.

You must also 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.

2.5 Restrictions

The following list identifies the limitations and restrictions on running the z/OS Batch Runtime:

- ▶ Updates to multiple databases are not supported; only one local database attachment is created and used by all applications running in the batch container.
- ▶ Only 31-bit EBCDIC applications are officially supported.
- ▶ A 31-bit version of the Java Virtual Machine (JVM) must be used.
- ▶ Multi-threaded Java DB2 applications are not supported.
- ▶ Java `System.exit` and COBOL `STOP BACK` calls cannot be used because they prevent the z/OS Batch Runtime from gaining control of the thread/TCB. Use `GOBACK` instead in your COBOL applications.
- ▶ COBOL applications must not code RRS Attach Facility (RRSAF) calls to initialize or end a DB2 connection. This can affect the DB2 connection managed by the z/OS Batch Runtime.
- ▶ There is no support for multiple resource managers; currently only JDBC is available.

2.6 Design considerations

In this section, we discuss several issues to consider when migrating or extending a COBOL application with Java code in this batch container.

2.6.1 Dynamic SQL versus static SQL

SQL statements are categorized as either *static* or *dynamic*. The difference between the two is that the syntax of a static SQL statement is fully known at precompile time, whereas the dynamic SQL statement syntax is not known until run time.

When compiling and linking an application with embedded static SQL statements, you must precompile and bind it. The precompiler converts the SQL statements into a form that the database manager understands and stores them into a *bind file*. The binding process creates a *package* from a bind file that the database manager uses to access the database when the application is run.

You might come across cases where you do not have all the necessary information to construct your SQL statements at precompile time. Dynamically constructing your SQL statements at run time allows you to reference objects that might not exist at run time, use the most optimal access path based on current database statistics, or experiment with special registers.

When designing your application for optimal performance, you must carefully decide whether to use static or dynamic SQL statements. The intended usage and working environment generally dictates this choice. The execution time of both these types of statements should be equivalent after they are compiled. Applications using dynamic SQL statements will require a higher initial cost per SQL statement because they need to be compiled before use. Despite the higher cost, dynamic SQL statements in some cases can run faster due to better access plans being chosen by the optimizer. The cost of compiling dynamic SQL statements can also vary because the statements might be implicitly recompiled by the system while the application is running.

Sometimes there is no obvious decision and it is best to choose the method you are most comfortable with. Table 2-1 is extracted from *IBM DB2 Universal Database Application Development Guide: Programming Client Applications*, SC09-4826 and provides guidelines for choosing which approach to use.

Table 2-1 Method to select SQL approach

Consideration	Likely Best Choice
Time to run the SQL statement <ul style="list-style-type: none"> ▶ Less than 2 seconds ▶ 2 to 10 seconds ▶ More than 10 seconds 	<ul style="list-style-type: none"> ▶ Static ▶ Either ▶ Dynamic
Data Uniformity <ul style="list-style-type: none"> ▶ Uniform data distribution ▶ Slight non-uniformity ▶ Highly non-uniform distribution 	<ul style="list-style-type: none"> ▶ Static ▶ Either ▶ Dynamic
Range (<,>,BETWEEN,LIKE) Predicates <ul style="list-style-type: none"> ▶ Very Infrequent ▶ Occasional ▶ Frequent 	<ul style="list-style-type: none"> ▶ Static ▶ Either ▶ Dynamic
Repetitious Execution <ul style="list-style-type: none"> ▶ Runs many times (10 or more times) ▶ Runs a few times (less than 10 times) ▶ Runs once 	<ul style="list-style-type: none"> ▶ Either ▶ Either ▶ Static
Nature of Query <ul style="list-style-type: none"> ▶ Random ▶ Permanent 	<ul style="list-style-type: none"> ▶ Dynamic ▶ Either
Run Time Environment (DML/DDDL) <ul style="list-style-type: none"> ▶ Transaction Processing (DML Only) ▶ Mixed (DML and DDL - DDL affects packages) ▶ Mixed (DML and DDL - DDL does not affect packages) 	<ul style="list-style-type: none"> ▶ Either ▶ Dynamic ▶ Either
Frequency of RUNSTATS <ul style="list-style-type: none"> ▶ Very infrequently ▶ Regularly ▶ Frequently 	<ul style="list-style-type: none"> ▶ Static ▶ Either ▶ Dynamic

2.6.2 Data recovery

Software can fail. In the event that your applications or the DB2 subsystem terminate abnormally, you need to ensure the integrity of your data. To design your batch database application for recovery:

1. Place all changes that must be made at the same time in the same unit of work. This ensures that the data remains in a consistent state if your applications or the DB2 subsystem terminate abnormally. Be sure to use the commit/rollback helper methods that the z/OS Batch Runtime offers, as described in “Commit rollback services” on page 10.
2. Consider how often to commit changes. If your application terminates abnormally, z/OS Batch Runtime backs out all uncommitted data changes, as discussed in “Cleanup processing” on page 11.
3. Consider how exceptions and abends are handled because DB2 will not intercept them from your application. As stated in “Cleanup processing” on page 11, any unhandled exceptions or abends cause the z/OS Batch Runtime to roll back all outstanding transactions. If this is not the desired action, then you are required to handle them to your expectations. Refer to “Handling return codes and Java exceptions” on page 9 for details.

2.6.3 JDBC Type 2

The z/OS Batch Runtime uses the JDBC Type 2 driver, implemented from a combination of Java and native code that converts JDBC method calls to the database client-side libraries. The native method calls yield better performance than the Java-implemented Type 3 and Type 4 JDBC drivers. You need to be mindful of the following limitations when designing applications that use the Type 2 JDBC driver:

- ▶ Database client-side libraries must be installed on the client system.
- ▶ Not all databases have a client-side library.
- ▶ The driver is platform dependent.
- ▶ The driver does not support Java Applets.

As mentioned in “Sharing a DB2 connection” on page 10, during startup, the z/OS Batch Runtime establishes a connection with the specified local database so that the applications running in the container can communicate with that database without having to create their own new connection.

Your Java applications must use the `Connection` class in the `java.sql` package and pass the URL String `jdbc:default:connection` to get the database connection (Example 2-11).

Example 2-11 Connection class in java.sql package

```
Connection connection = DriverManager.getConnection("jdbc:default:connection");
```

2.7 Migrating a hybrid COBOL/Java application

The reason to migrate your existing hybrid database application to this batch container is to take advantage of the single database connection sharing capability so that you can run your multi-component spanning database changes as a single unit of work.

Consider the following questions when planning your migration:

- ▶ Do you have any database transactions that span across your COBOL and Java applications?
- ▶ Will your hybrid application be affected by the limitations of this release? See 2.3, “Required software” on page 11. For example, only a single local database can be established in the container, so if your applications communicate with multiple or remote databases, this container might not be ideal.

If this batch container meets your requirements and your applications are not affected by its limitations, then the following simple code changes are required:

- ▶ Remove all thread/TCB terminating calls – `System.exit` and `STOP BACK`, see “Terminating managed applications” on page 10.
- ▶ In your Java code, ensure the `DriverManager.getConnections` method uses the URL `jdbc:default:connection` to get the existing connection to the local database system.
- ▶ Write and configure the z/OS Batch Runtime options in your JCL to invoke the application and the container.



End-to-end development scenario

In this chapter, we describe the development scenario we used to extend a batch COBOL application with Java code to illustrate the design considerations and usage of the z/OS batch runtime container. In our scenario, we used IBM Rational Developer for System z (RDz) to demonstrate how modern tooling can be used to extend legacy batch applications, increase productivity, and accelerate mainframe application development.

We also demonstrate how to extend an existing stand-alone batch DB2 COBOL application with new Java functionality. Samples of the source code and JCL are provided to illustrate the changes required to run the modified COBOL application in z/OS Batch Runtime.

We used a step-by-step format and included screen shots that should be helpful for users who might not be familiar with Integrated Development Environments (IDEs) such as Eclipse.

3.1 Software used in the scenario

We used the following software levels for our scenario:

- ▶ z/OS V1.13
- ▶ IBM Rational Developer for System z with EGL 8.0.1
- ▶ IBM 64-bit SDK for z/OS, Java Technology Edition, V6
- ▶ Enterprise COBOL for z/OS Version 4 Release 2 Modification 0
- ▶ DB2 9 for z/OS

3.2 Infrastructure setup

In “Invoking z/OS Batch Runtime” on page 11 we reviewed how to invoke z/OS Batch Runtime through a batch JCL. In this section we show how to verify the installation of Java and the z/OS Batch Runtime.

3.2.1 z/OS Batch Runtime verification

To verify the z/OS Batch Runtime installation, run the verification program *BCDIVP* as shown in Example 3-1.

Example 3-1 Minimal version of BCDIVP

```
//BCDIVP JOB 'ZAID FAYDI',CLASS=A,MSGLEVEL=(1,1),
// MSGCLASS=T,NOTIFY=&SYSUID
//*
//BATCH EXEC BCDPROC,REGION=OM,LOGLVL='+I'
//*****
/* UPDATE: If the jZOS Java launcher has not been installed in      *
/*          the lnk1st, add a steplib for it.                       *
//*****
/*STEPLIB DD DSN=h1q.jzos.loadlib,DISP=SHR
//STEPLIB DD DISP=SHR,DSN=DBOVT.SDSNLOAD
//          DD DISP=SHR,DSN=DBOVT.SDSNLOD2
//*
//STDENV DD *
#-----
# UPDATE: Installation path for batch runtime.
#-----
export BCD_HOME=/usr/lpp/bcp
#-----
# UPDATE: Installation path for Java.
#-----
export JAVA_HOME=/usr/lpp/java/J6.0.1
#-----
# The following runs the batch runtime configuration script.
# This script processes the exported container variables that
# were defined above.
#-----
. $BCD_HOME/bcdconfig.sh
#-----
```

```

# UPDATE: Uncomment and update JDBC driver files.
#-----
JDBC_HOME=/usr/lpp/db2/db0v/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"
#-----
# UPDATE: Add any JDBC options here. See the DB2 Information
# Center for details on the options.
#-----
IJO="$IJO -Ddb2.jcc.ssid=DB0V"
#-----
# Exports JVM options set above.
#-----
export IBM_JAVA_OPTIONS="$IJO "
#-----
# The following runs the batch runtime configuration completion
# script. This command must be last in the STDENV file.
#-----
. $BCD_HOME/bcdconfigend.sh
//*****
//* z/OS Batch Runtime Options *
//*****
//BCDIN DD *
#-----*
# The following sets the language and class name for the IVP program.
#-----*
bcd.applicationLanguage=JAVA
bcd.applicationName=com.ibm.zos.batch.container.BCDAbout
#-----*
# UPDATE: Support class name used to manage transactions.
#-----*
bcd.supportClass.1=com.ibm.db2.jcc.t2zos.T2zosBatchcontainerSupport
#-----*
//

```

STDOUT should contain a report similar to what is displayed in Example 3-2.

Example 3-2 Snippets of the BCPIVP report

```

...
Product Information:
-----
Product Name: z/OS V1R13
Product FMID: HBB7780
Product number: 5694-A01
Product component ID: 5752SC1BC
Component name: Batch Runtime
Build ID: 2011011_130751701_mt19483
Framework ID: BATCC10.BATCH [cf011104.03]
Java Environment:
-----

```

```

Java version: 1.6.0
Java vm name: IBM J9 VM
Java library path: /usr/lpp/java/J6.0.1/lib/s390/default: ...
...
Class Levels in /Z1DRC1/usr/lpp/bcp/batch/bcdbatch.jar:
-----
HBB7780  com.ibm.zos.batch.container.BCDAbout
HBB7780  com.ibm.zos.batch.container.BCDBatchcontainer
HBB7780  com.ibm.zos.batch.container.config.BCDConfig
...
HBB7780  com.ibm.zos.batch.container.util.BCDMessage
HBB7780  com.ibm.zos.batch.container.util.BCDUtil
-----
End of Report

```

3.2.2 Java

The output from the BCPIVP program shows the level of Java installed.

However, if you are having difficulties running the JCL and want to do some troubleshooting, this is another way to check whether the required version and level of Java is installed on your system:

- ▶ Enter the OMVS shell
- ▶ From ISPF, enter **TSO OMVS**
- ▶ At the console, enter **java -version**

The output should be similar to Example 3-3.

Example 3-3 Sample Java version output

```

FAYDI:/u/faydi: >java -version
java version "1.6.0"
Java(TM) SE Runtime Environment (build pmz3160sr9fp1-20110303_01(SR9 FP1))
IBM J9 VM (build 2.4, JRE 1.6.0 IBM J9 2.4 z/OS s390-31
jvmmz3160sr9-20110203_74623 (JIT enabled, AOT enabled)
J9VM - 20110203_074623
JIT - r9_20101028_17488ifx3
GC - 20101027_AA)
JCL - 20110203_01

```

To download and learn more about z/OS Java Technology Edition, refer to:

http://www-03.ibm.com/systems/z/os/zos/tools/java/products/sdk601_31.html

3.3 Setting up the development environment

In this section we outline the steps to create all the sub-projects, data sets, files, and tables for the development scenario using Rational Developer for System z.

3.3.1 Connecting to z/OS from Rational Developer for System z

1. Start Rational Developer for System z. In the z/OS Project Perspective, click the **New Connection** button in the Remote Systems view (Figure 3-1).

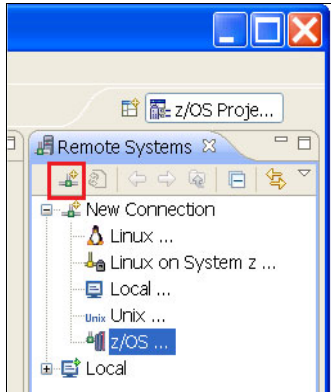


Figure 3-1 Rational Developer: Remote system view

2. Select **z/OS** as the System type and click **Next** (Figure 3-2).

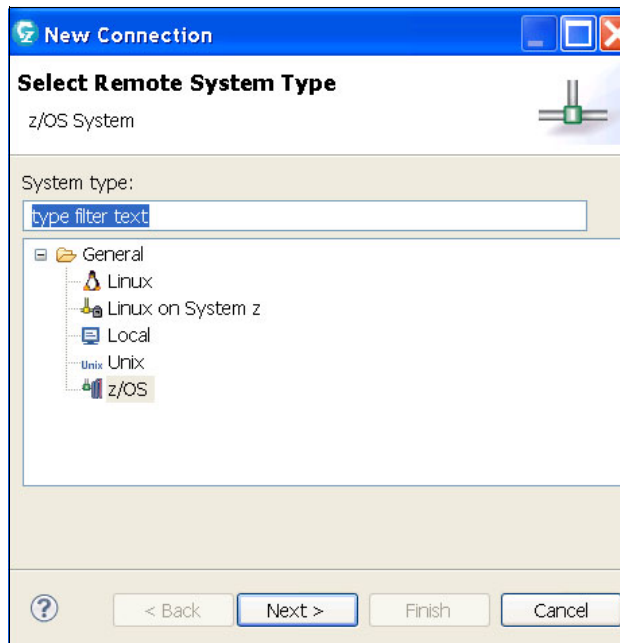


Figure 3-2 New connection: Select remote system type

3. Specify the host name of your z/OS system running the RDz server, then click **Next** (Figure 3-3).

The screenshot shows a window titled "New Connection" with a blue header bar. Below the header, the title "Remote z/OS System Connection" is displayed in bold, followed by the subtitle "Define connection information". The main area contains several input fields: "Parent profile:" with a dropdown menu showing "faydi"; "Host name:" with a dropdown menu showing "wtsc64.itso.ibm.com"; "Connection name:" with a text box containing "wtsc64.itso.ibm.com"; and "Description:" with an empty text box. Below these fields is a checkbox labeled "Verify host name" which is checked. At the bottom of the window, there are four buttons: a help icon (?), "< Back", "Next >", "Finish", and "Cancel".

Figure 3-3 New connection: Specify host name

4. Specify the z/OS UNIX subsystem information, as customized by your systems programmer, then click **Next** (Figure 3-4).

The screenshot shows a window titled "New Connection" with a blue header bar. Below the header, the title "z/OS UNIX Files" is displayed in bold, followed by the subtitle "Define subsystem information". The main area contains a section titled "Indicate how the remote server should be launched by default:" with a radio button selected for "Remote daemon". Below this, there are two input fields: "Daemon Port (1-65535)" with the value "4045" and "Authentication method" with a dropdown menu showing "userid/password".

Figure 3-4 New connection: Define z/OS UNIX subsystem information

5. Specify the IBM MVS™ subsystem information, as customized by your systems programmer, then click **Finish** (Figure 3-5).

The screenshot shows a window titled "New Connection" with a blue header bar. Below the header, the title "MVS Files" is displayed in bold, followed by the subtitle "Define subsystem information". The main area contains a section titled "Indicate how the remote server should be launched by default:" with a radio button selected for "Remote daemon". Below this, there are two input fields: "Daemon Port (1-65535)" with the value "4045" and "Authentication method" with a dropdown menu showing "userid/password".

Figure 3-5 New connection: Define MVS subsystem information

- To connect to the z/OS system, right-click the newly created node in Remote Systems and select **Connect** (Figure 3-6).

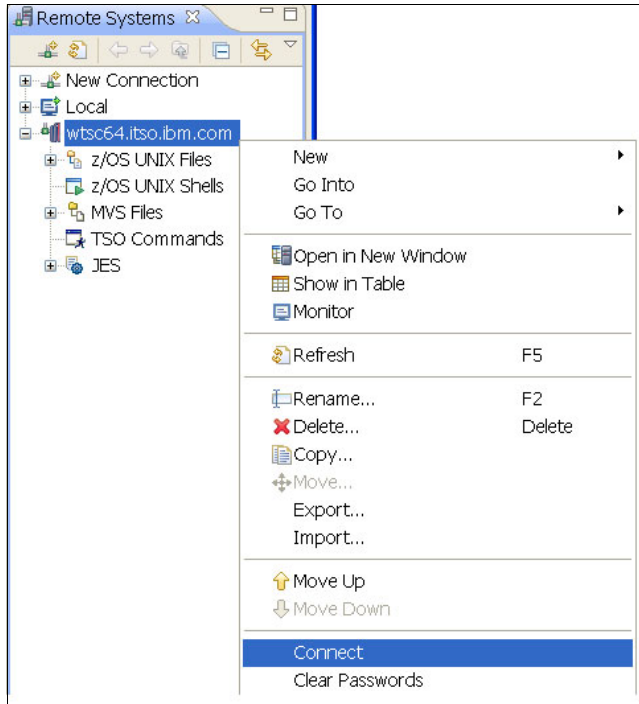


Figure 3-6 Connect to z/OS system

- If prompted with the window shown in Figure 3-7, click **Yes** to proceed.

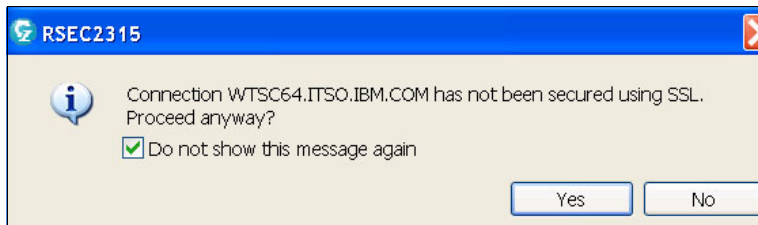


Figure 3-7 Connection prompt

- To confirm that you are connected and that your MVS and HFS subsystem information is correct, expand the z/OS UNIX Files and MVS Files nodes and their child nodes. Expand the JES node and confirm that a list of jobs is returned. If a list is not returned, then it might be that you have no jobs to retrieve or you need to set the correct JES Job Monitor port number, as shown in Figure 3-8. To correct the setting, right-click the JES node and select **Properties**.

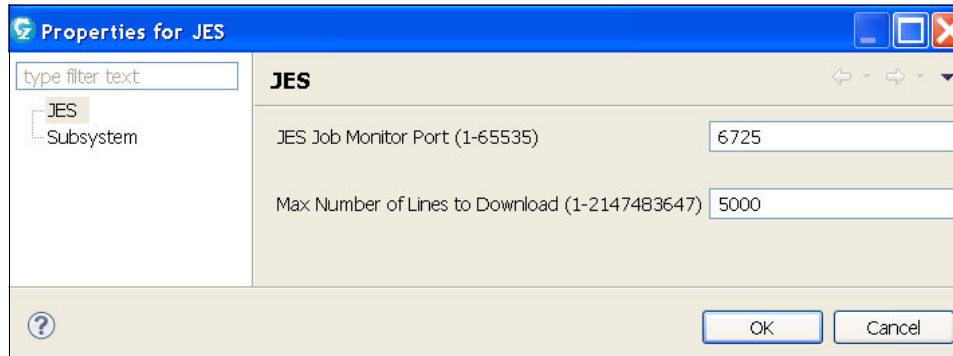


Figure 3-8 JES properties

3.3.2 Creating the HFS directories

Use the following steps to create the required HFS directories.

1. In the z/OS UNIX Files directory, right-click the **My Home** filter and select **New** → **Folder** (Figure 3-9).

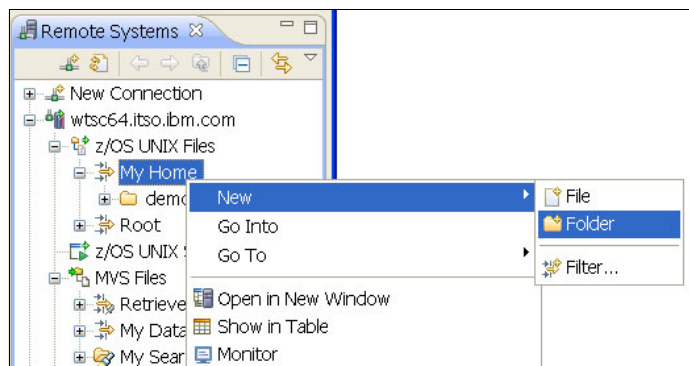


Figure 3-9 Remote systems - z/OS UNIX files

2. Enter com as the folder name and click **Finish** (Figure 3-10).

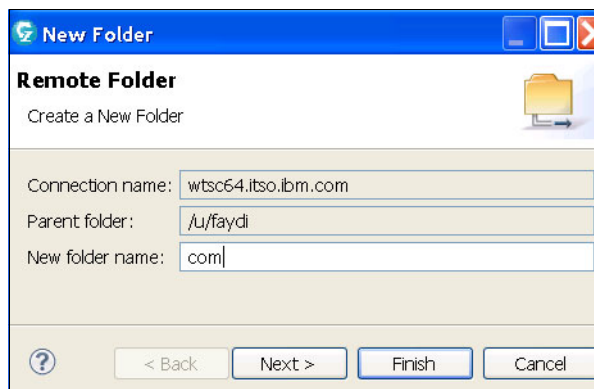


Figure 3-10 Create new folder

3. Create a folder called sample under the newly created com folder, then create lib, src and statements folders under sample, as shown in Figure 3-11 on page 25.

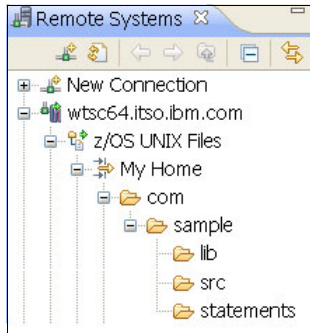


Figure 3-11 Create additional folders

3.3.3 Creating the z/OS project and MVS and UNIX subprojects

Use the following steps to create the z/OS project and MVS and UNIX subprojects.

1. In the z/OS Projects view, right-click and select **z/OS Project** (Figure 3-12).

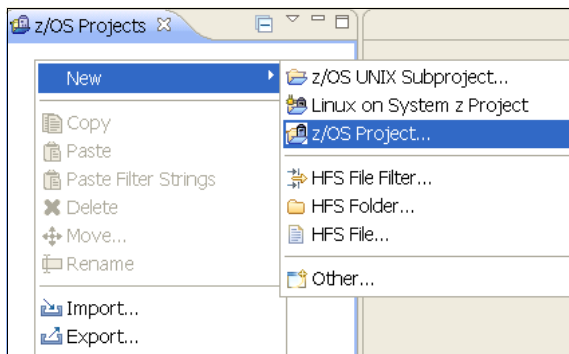


Figure 3-12 Select z/OS Project

2. Enter BatchProject as the Project name, ensure that the Create an MVS subproject radio button is selected, then click **Finish** (Figure 3-13).

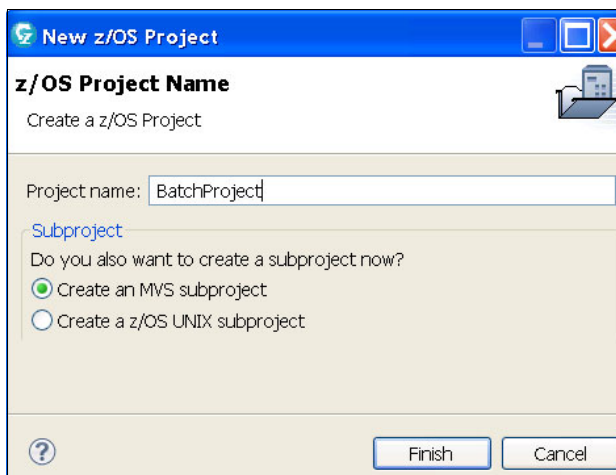


Figure 3-13 New z/OS project name

3. The New MVS Subproject window appears. Set the high-level qualifier field and the remaining fields as shown in Figure 3-14, then click **Finish**.

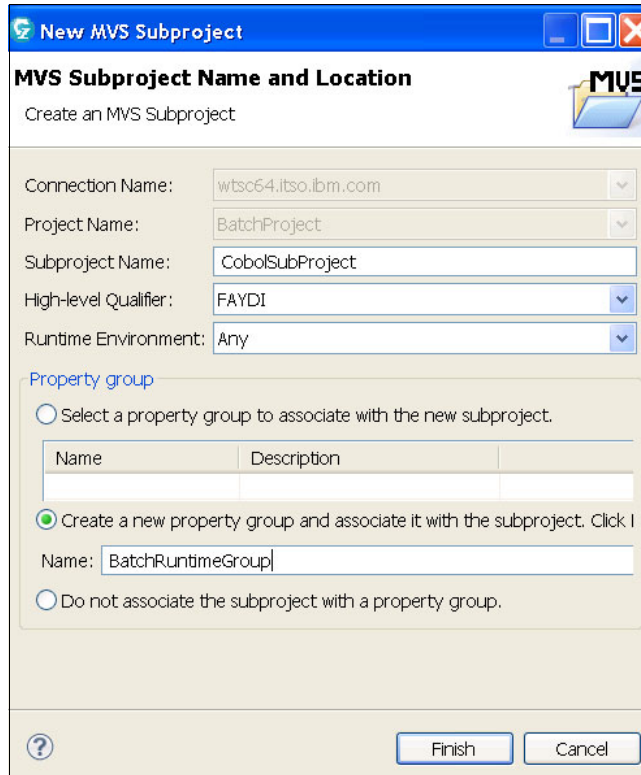


Figure 3-14 New MVS subproject name and location

4. You also need a UNIX subproject for your new Java code. To create a UNIX subproject, right-click the BatchProject node and select **New** → **z/OS UNIX Subproject**. The New z/OS UNIX Subproject wizard appears. Select **BatchProject** as the parent project and click **Next** (Figure 3-15).

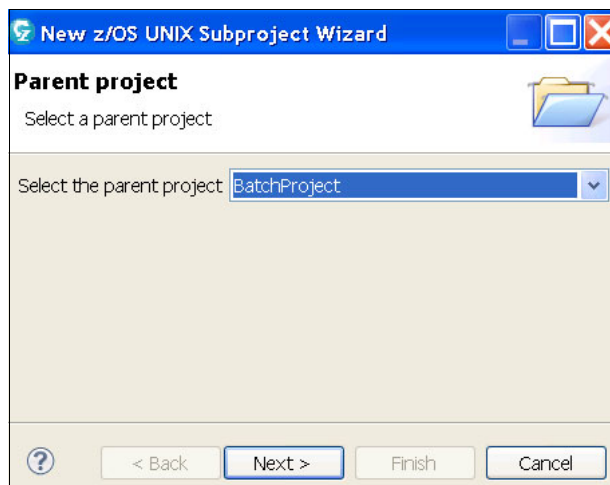


Figure 3-15 Parent project selection

5. Enter JavaSubProject as the subproject name (Figure 3-16).

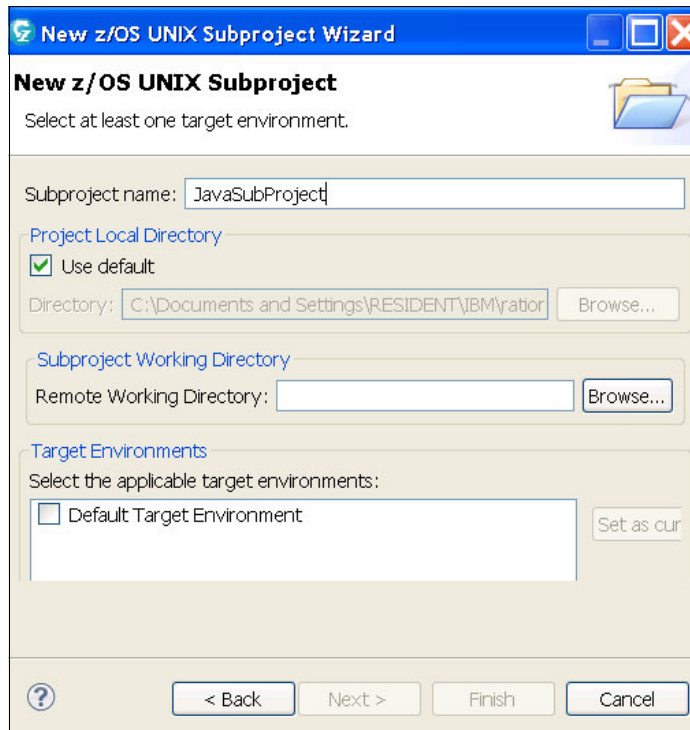


Figure 3-16 Enter new z/OS UNIX subproject name

6. Click the Remote Working Directory **Browse** button, navigate to the sample folder you created earlier, and click **OK** (Figure 3-17).

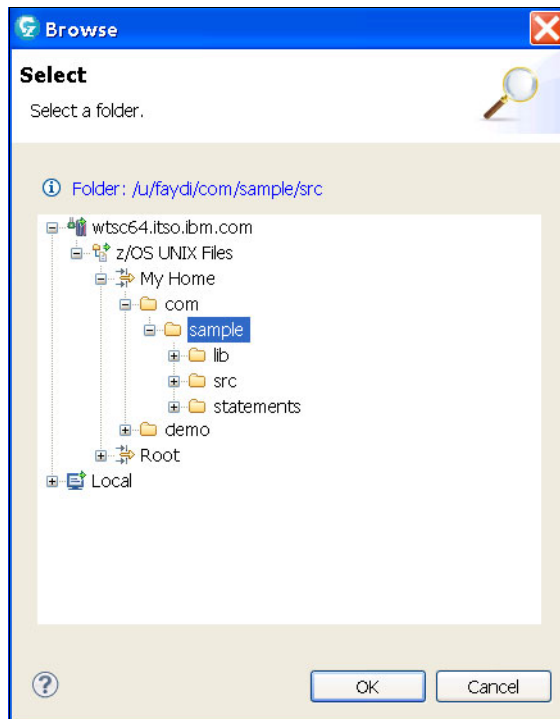


Figure 3-17 Select a folder

7. Select the Default Target Environment check box and click **Next** (Figure 3-18).

hi

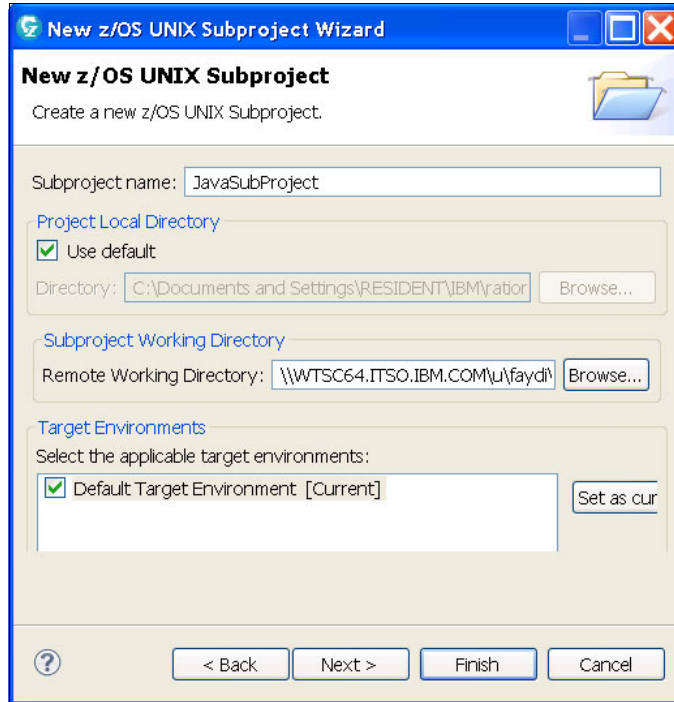


Figure 3-18 Create a new z/OS UNIX subproject

8. Specify a filter for this new subproject. Enter sample as the filter name and click **Next** (Figure 3-19).

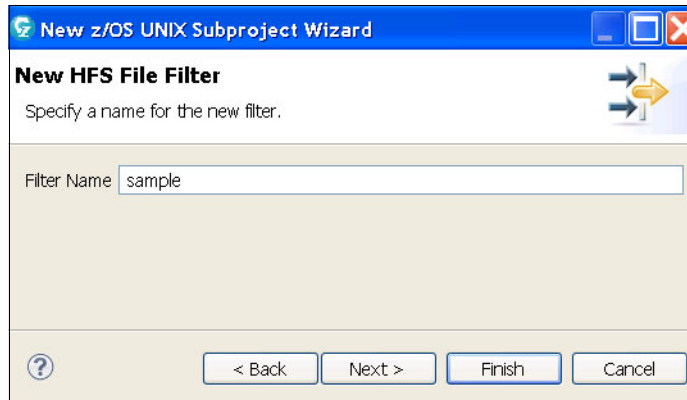


Figure 3-19 New HFS file filter

- To set this filter to only show the .java source files, click **Add** and enter * for the file name pattern, then click **OK** (Figure 3-20).

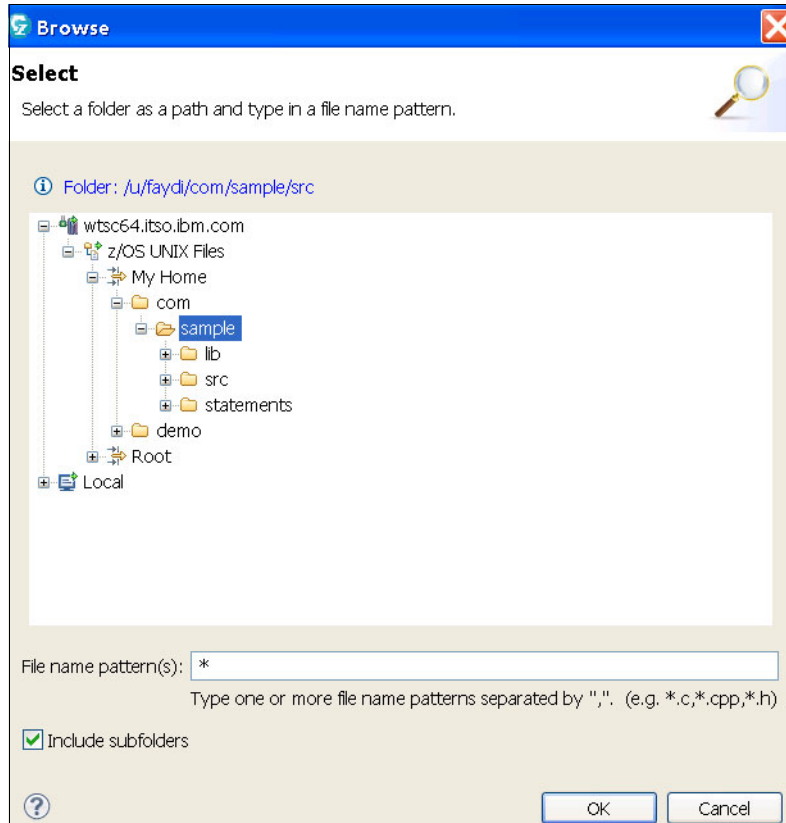


Figure 3-20 Select folder

- Click **Finish** to create the filter (Figure 3-21).

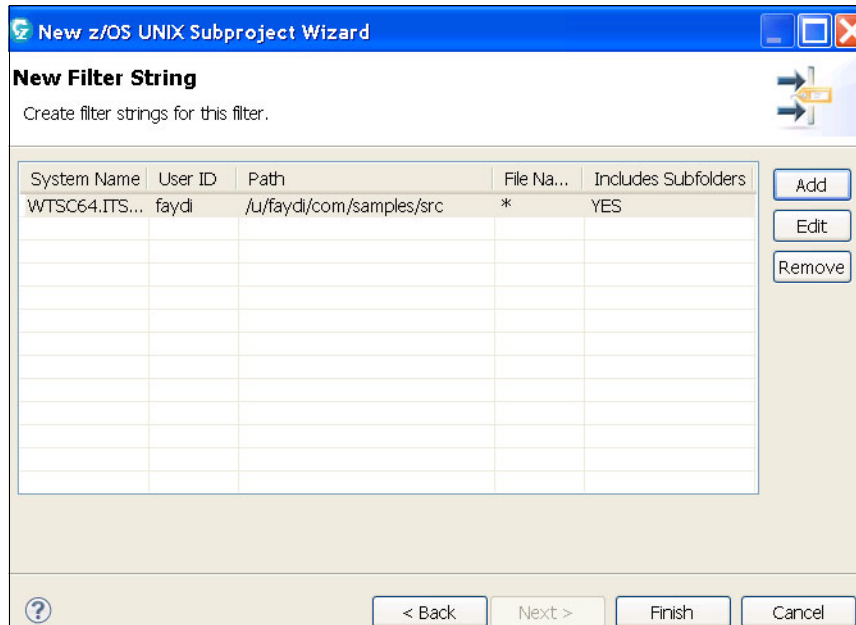


Figure 3-21 Create filter string

11. You now should have the z/OS Project, MVS subproject, and UNIX subproject created under the z/OS Projects view, as shown in Figure 3-22.

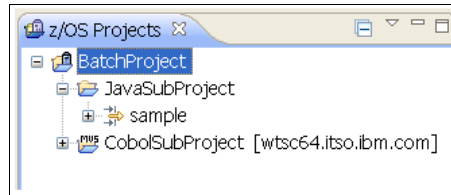


Figure 3-22 z/OS projects view

3.3.4 Creating the DB2 QUERIES table

Use the following steps to create a DB2 Queries table.

1. Open the Data perspective, click **window** → **Open Perspective** → **Other**, and select **Data** from the Open Perspective window (Figure 3-23).

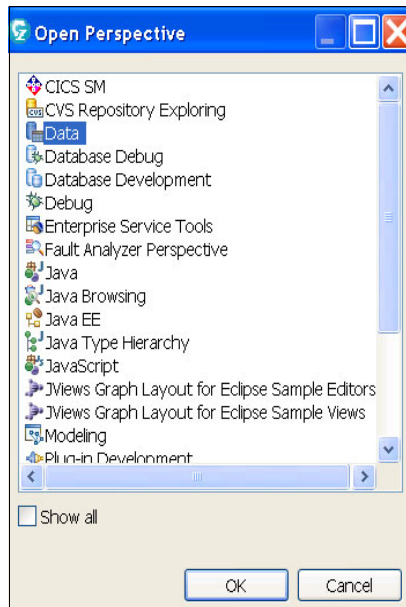


Figure 3-23 Open Perspective window

2. In the Data Source Explorer, click the New Connection button (Figure 3-24).

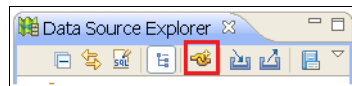


Figure 3-24 Data Source Explorer

3. Select the **DB2 for z/OS** database manager and populate the required fields: Location, Host, Port number, User name, and Password (Figure 3-25).

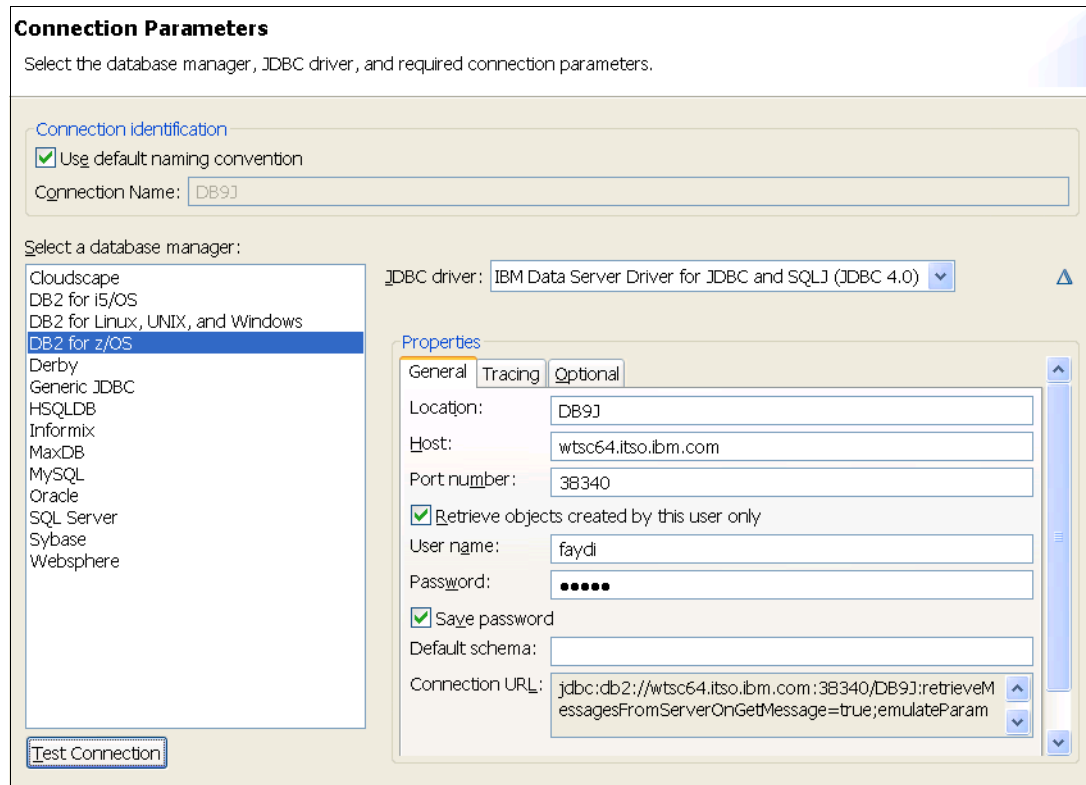


Figure 3-25 Connection Parameters

4. Click **Test Connection** and confirm that the New Connection window appears with the message Connection succeeded (Figure 3-26).

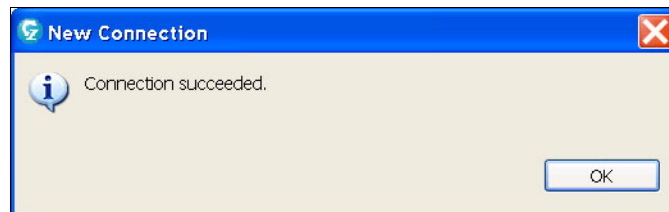


Figure 3-26 New connection window

5. Click **Finish**. The database connection should appear in the Data Source Explorer (Figure 3-27).

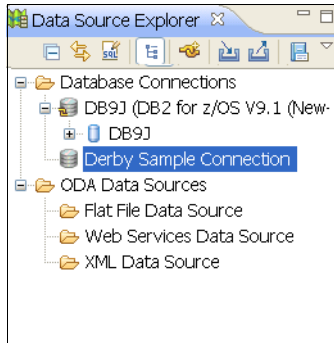


Figure 3-27 Data Source Explorer

6. Before you can add the SQL statements to create the sample table, you must create a new Data Set Project. Right-click in the Data Project Explorer and select **New** → **Data Design Project** (Figure 3-28).

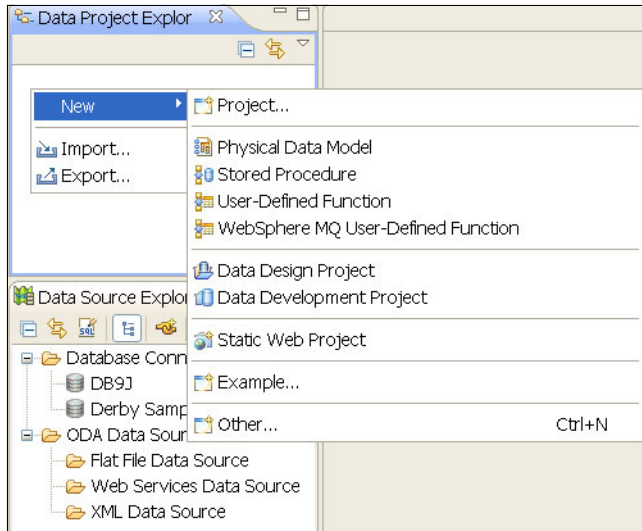


Figure 3-28 Data Project Explorer

7. Enter BatchProjectData as the project and click **Finish** (Figure 3-29).

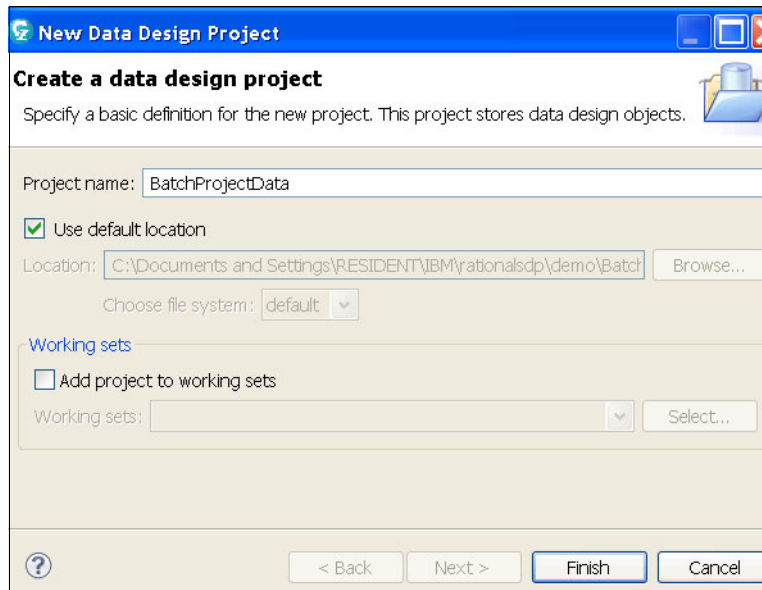


Figure 3-29 Create data design project

8. To create a new SQL Script, expand the newly created BatchProjectData node, right-click SQL Scripts, and select **New** → **SQL or XQuery Script** (Figure 3-30).

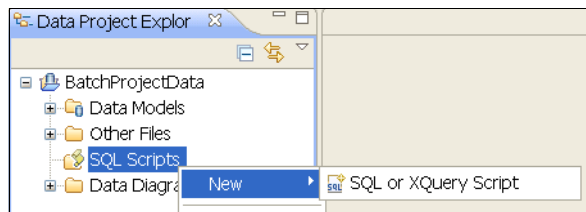


Figure 3-30 New SQL or XQuery Script

9. Name the SQL script CreateQueriesTable and click **Finish** (Figure 3-31).

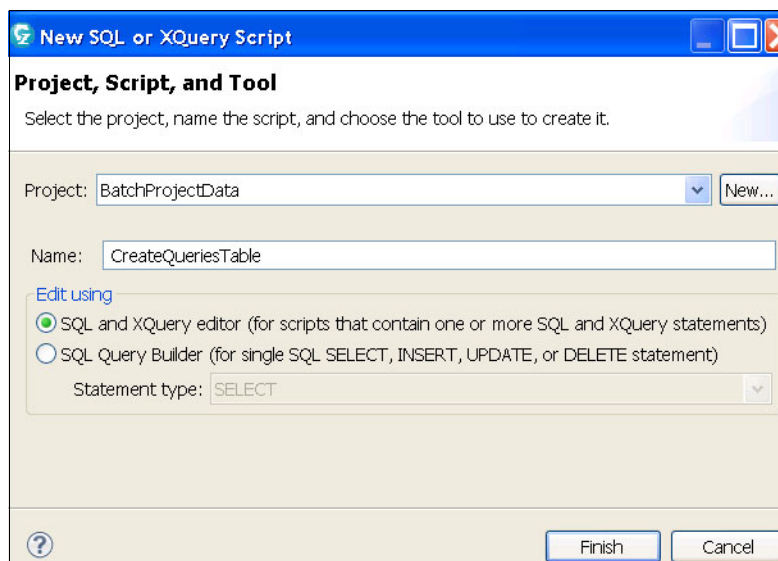


Figure 3-31 Select project and name script

10. In the Select Connection Profile window, select the database, in this case **DB9J**, and click **Finish** (Figure 3-32).

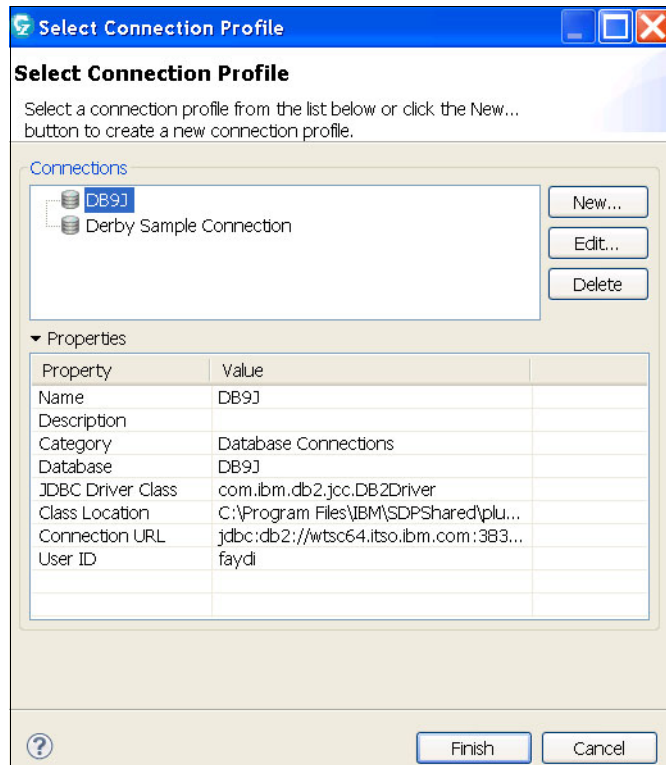


Figure 3-32 Select connection profile

11. An editor is opened. Copy the SQL statements shown in Example 3-4 into the editor.

Example 3-4 SQL statements

```
CREATE TABLE QUERIES (
    AMOUNT    DECIMAL(9,2)    NOT NULL,
    INTEREST  DECIMAL(4,2)    NOT NULL,
    PERIODS   INTEGER         NOT NULL,
    PAYMENT   VARCHAR(15)     NOT NULL,
    PATH      VARCHAR(100),
    PRIMARY KEY (AMOUNT, INTEREST, PERIODS)
);
```

12. Paste the SQL statements in **Data Project Explorer** → **CreateQueriesTable.sql**, as shown in Figure 3-33.

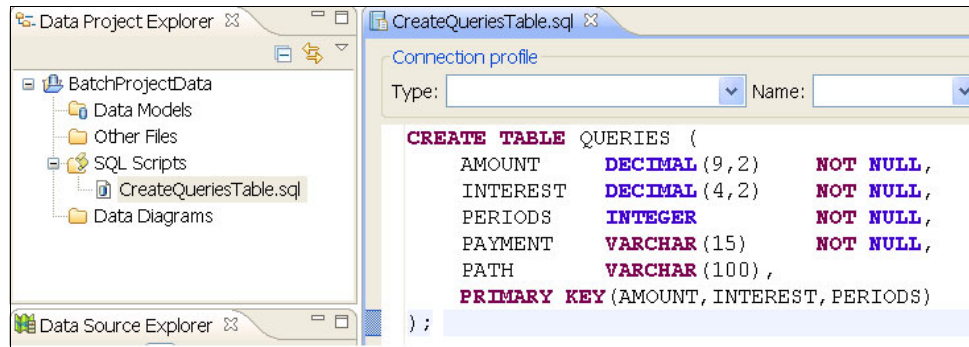


Figure 3-33 Data Project Explorer: CreateQueriesTable.sql

13. Right-click the newly created CreateQueriesTable.sql SQL script and click **Run SQL** (Figure 3-34).

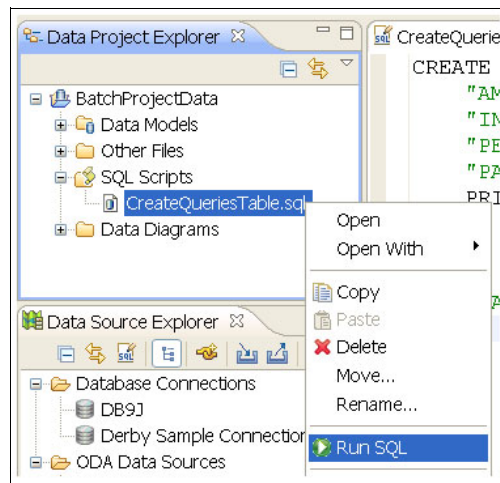


Figure 3-34 Run SQL

14. In the Select Connection Profile window, select **DB9J** and click **Finish** (Figure 3-35).

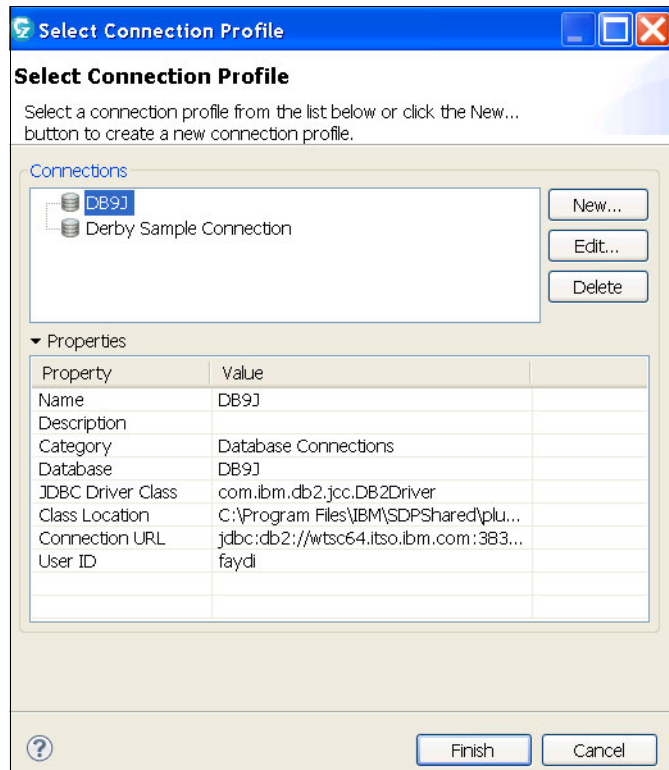


Figure 3-35 Select Connection Profile

15. In the SQL Results view you should see a successful execution and the table created under the **Schemas** → **SQLID** → **Tables** node in the Data Source Explorer view (Figure 3-36).

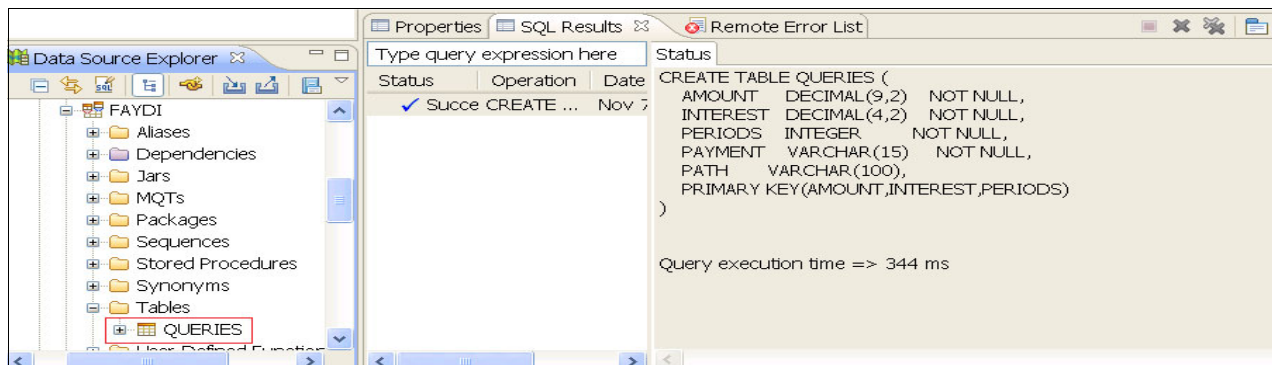


Figure 3-36 Data Source Explorer view

16. Right-click the **QUERIES** table and select **Data** → **Edit** (Figure 3-37).

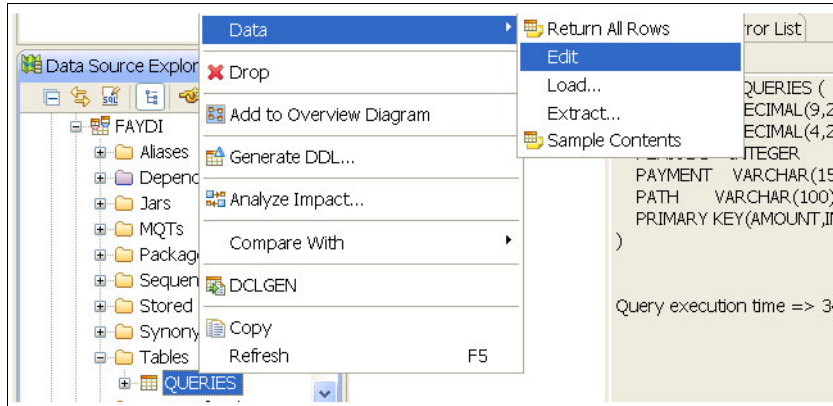


Figure 3-37 Data Source Explorer: **QUERIES** table

17. From this editor, you can insert or delete records and set field data; for now, leave it as is (Figure 3-38).

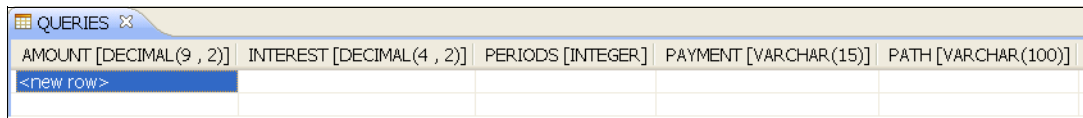


Figure 3-38 Edit **QUERIES**

3.3.5 Creating sample data sets

We provide these steps to illustrate how RDz can be used to create the different types of data sets without requiring the user to enter all the necessary details, such as record length and organization.

1. Create a filter to display only the created data sets. Right-click the **My Data Sets** node and select **New** → **Filters** (Figure 3-39).

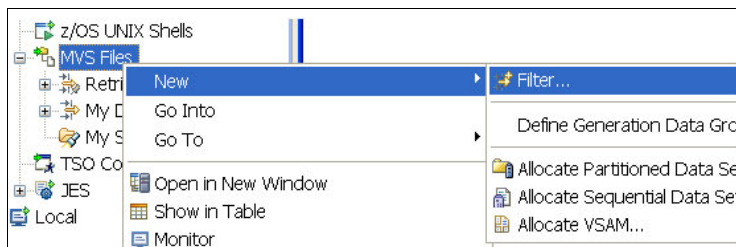


Figure 3-39 My Data Sets: New filter

2. Enter `<HLQ>.ITS0.*` as the filter string and click **Next** (Figure 3-40). *HLQ* is your high-level qualifier.

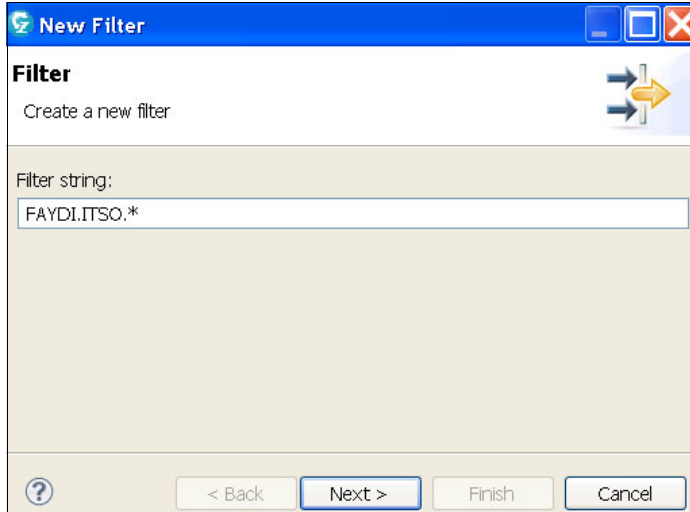


Figure 3-40 Create new filter

3. Enter `ITS0` as the filter name and click **Finish** (Figure 3-41).

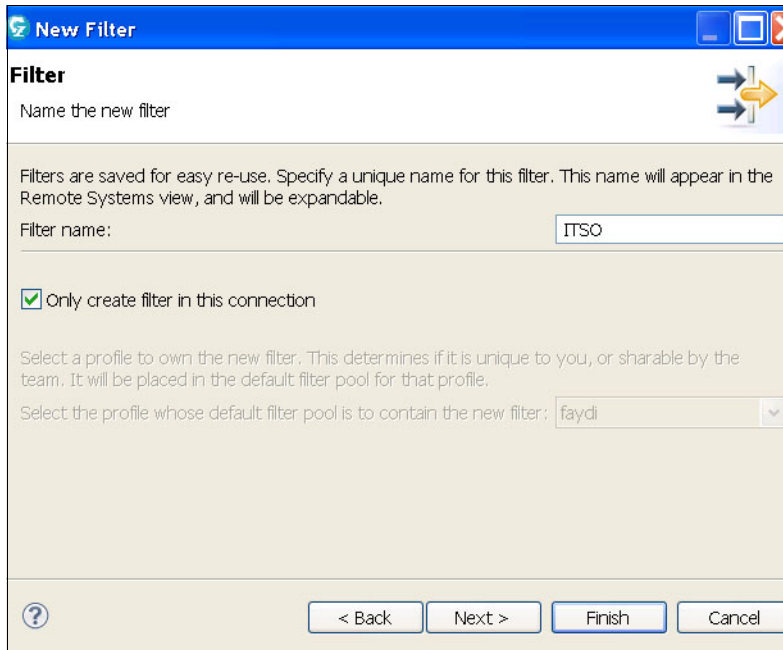


Figure 3-41 Name the new filter

4. You should now have the ITSO filter with no matching data sets (Figure 3-42).

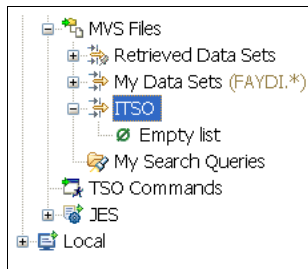


Figure 3-42 ITSO filter with no matching data sets

5. To allocate data sets, right-click **CobolSubProject** in the z/OS Projects view and select **New** → **Allocate Partitioned Data Set** (Figure 3-43).

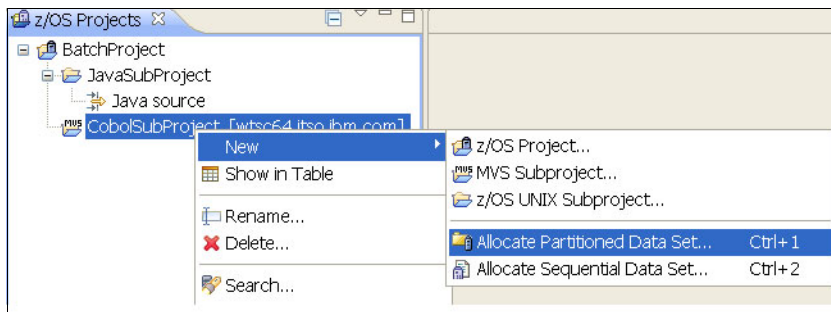


Figure 3-43 New Allocate Partitioned Data Set

6. Enter ITSO.COBOL in the Data Set Name textbox and click **Next** (Figure 3-44).

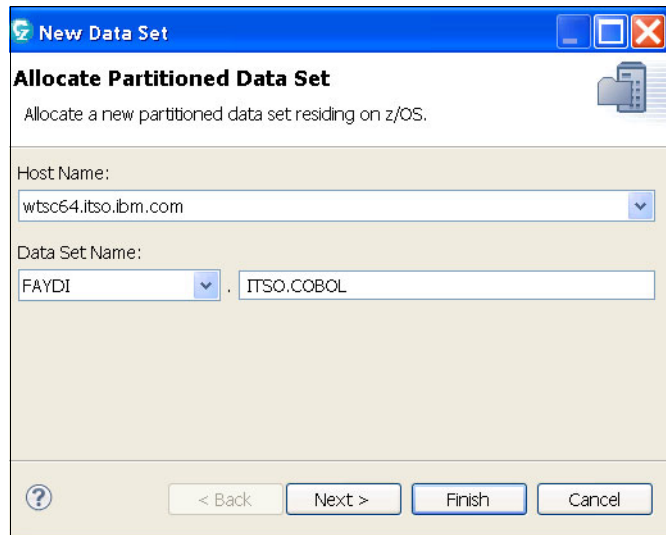


Figure 3-44 Allocate partitioned data set on z/OS

7. Select Category SOURCE and Type COBOL, then click **Finish** (Figure 3-45).

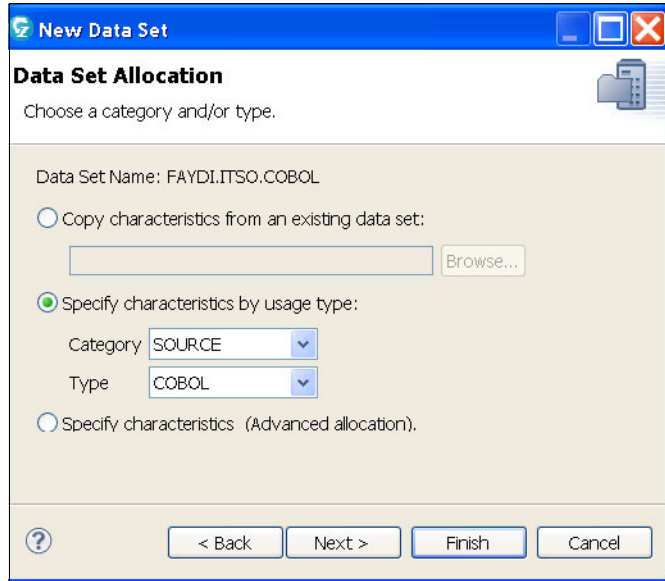


Figure 3-45 Choose data set category

8. The <HLQ>.ITSO.COBOL data set is now allocated and should appear under CobolSubProject and the ITSO filter under MVS Files in the Remote Systems view, created earlier (Figure 3-46).

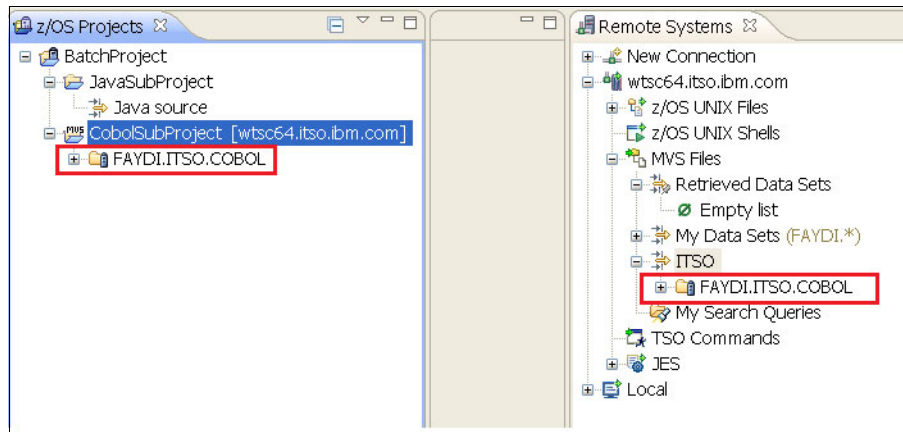


Figure 3-46 View of ITSO.COBOL data set

9. Repeat steps 1 through 8 to create the following data sets:

- <HLQ>.ITSO.JCL - Category: SOURCE, Type: JCL
- <HLQ>.ITSO.LOAD - Category: Others, Type: LOADMOD
- <HLQ>.ITSO.COBYLIB - Category: SOURCE, Type: COBOL
- <HLQ>.ITSO.OBJLIB - Category: Others, Type: OBJECT
- <HLQ>.ITSO.LISTING - Category: LISTING, Type: COBOL
- <HLQ>.ITSO.DBRLM - Category: Others, Type: DBRLM

3.4 Developing COBOL, Java, and JCL

In this section, we provide all the steps and source code to create COBLOAN, PdfCreator, and the necessary JCL to compile, link, bind, and execute the application.

COBLOAN is a simple stand-alone batch COBOL application that calculates the monthly repayments of a loan given the principle amount, fixed interest rate, and number of periods the interest is charged. The amount is displayed in the job output and all values are inserted into a DB2 table for record keeping.

PdfCreator is a Java class that implements a new business requirement to generate Portable Document Format (PDF) files of each calculation. The path of the PDF file generated is stored in the same table COBLOAN uses. The two database transactions from COBLOAN and PdfCreator are set to be a single unit of work. If either transaction fails, both changes are discarded.

For your convenience, we have provided the source files for this scenario. If you want to reproduce the development scenario without having to copy/paste the source code out of this chapter, you can FTP the files described in “Using the Web material” on page 63 to your system.

3.4.1 Adding COBLOAN

Use the following steps to add COBLOAN.

1. Right-click <HLQ>.ITSO.COBL and select **New** → **Create Member** (Figure 3-47).

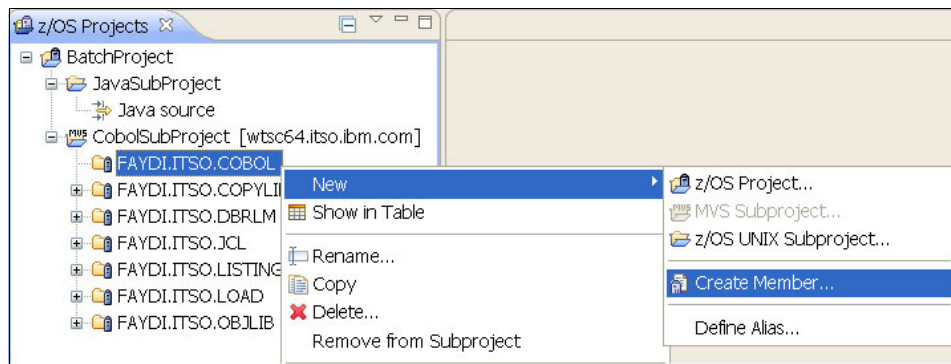


Figure 3-47 z/OS Projects - Create Member

2. Enter COBLOAN into the Member Name textbox and click **Finish** (Figure 3-48).

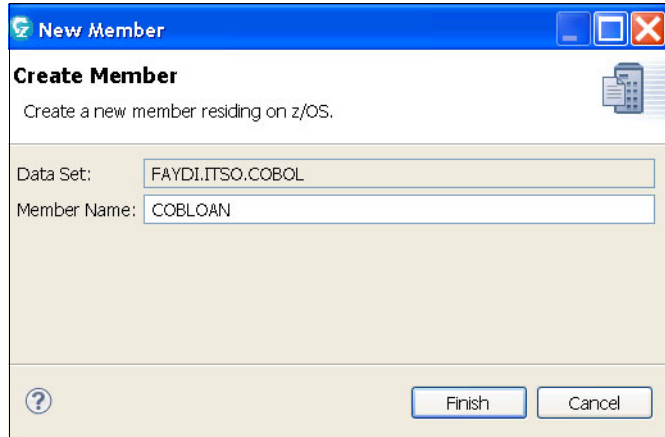


Figure 3-48 Enter new Member Name

3. COBLOAN will be created under <HLQ>.ITSO.COBOL with a .cbl extension, which is not set on the host. Double-click this new member to invoke the editor and copy the COBOL code shown in Example 3-5.

Example 3-5 Sample COBOL code

```

*****
* COBLOAN *
* * *
* A simple program that calculates payment amount for a *
* loan, stores the calculation into a DB2 table, and *
* generates a PDF file containing the calculation. *
* * *
* Example input: '30000 .09 24 ' *
* * *
*****

IDENTIFICATION DIVISION.
PROGRAM-ID. 'COBLOAN' RECURSIVE.
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
REPOSITORY.
    Class PdfCreator      is "PdfCreator"
    Class jstring         is "jstring".

DATA DIVISION.
WORKING-STORAGE SECTION.
01 FIELDS.
    05 PAYMENT            PIC S9(9)V99 USAGE COMP.
    05 PAYMENT-OUT        PIC $$$,$$$,$$9.99 USAGE DISPLAY.
    05 LOAN-AMOUNT        PIC S9(7)V99 USAGE COMP.
    05 LOAN-AMOUNT-IN     PIC X(16).
    05 INTEREST-IN        PIC X(5).
    05 INTEREST           PIC S9(3)V99 USAGE COMP.
    05 NO-OF-PERIODS-IN   PIC X(2).
    05 NO-OF-PERIODS      PIC 99 USAGE COMP.
01 RC                    PIC S9(9) COMP-5.
01 parmString            OBJECT REFERENCE jstring.

```

```

01 PAYMENT-SQL          PIC X(15).
01 STRING-BUFFER       PIC X(42).
01 A-SQLCODE           PIC S9(3) SIGN IS LEADING SEPARATE.

EXEC SQL INCLUDE SQLCA END-EXEC.

LINKAGE SECTION.
01 INPUT-PARM.
    05 INPUT-LENGTH     PIC S9(4) COMP.
    05 INPUT-VALUE      PIC X(26).
COPY JNI.

PROCEDURE DIVISION USING INPUT-PARM.
    SET ADDRESS OF JNIENV TO JNIENVPTR
    SET ADDRESS OF JNINATIVEINTERFACE TO JNIENV

* Parse input
    UNSTRING INPUT-VALUE DELIMITED BY ALL " "
        INTO LOAN-AMOUNT-IN INTEREST-IN NO-OF-PERIODS-IN.
* Convert to numeric values
    COMPUTE LOAN-AMOUNT = FUNCTION NUMVAL(LOAN-AMOUNT-IN).
    COMPUTE INTEREST = FUNCTION NUMVAL(INTEREST-IN).
    COMPUTE NO-OF-PERIODS = FUNCTION NUMVAL(NO-OF-PERIODS-IN).
* Calculate annuity amount required
    COMPUTE PAYMENT = LOAN-AMOUNT *
        FUNCTION ANNUITY((INTEREST / 12 ) NO-OF-PERIODS).
* Format then display payment
    MOVE PAYMENT TO PAYMENT-OUT.
    DISPLAY 'Payment ' PAYMENT-OUT.
    MOVE PAYMENT-OUT TO PAYMENT-SQL.

    DISPLAY LOAN-AMOUNT-IN ' '
        INTEREST-IN ' '
        NO-OF-PERIODS-IN ' '
        PAYMENT-SQL.
* Insert calculate into a table
    EXEC SQL
        INSERT INTO INQUIRIES (
            AMOUNT,
            INTEREST,
            PERIODS,
            PAYMENT
        )
        VALUES (
            DECIMAL(:LOAN-AMOUNT-IN,9,2),
            DECIMAL(:INTEREST-IN,4,2),
            INTEGER(:NO-OF-PERIODS-IN),
            :PAYMENT-SQL
        )
    END-EXEC.
    IF SQLCODE NOT EQUAL ZERO
        MOVE SQLCODE TO A-SQLCODE
        DISPLAY 'Insert record failed ' A-SQLCODE ' ' SQLSTATE
    ELSE
        DISPLAY 'insert record successful'.

```

```

* Create String Object of the calculations to pass to Java
STRING
    PAYMENT-OUT DELIMITED BY SIZE
    ' ' DELIMITED BY SIZE
    INPUT-VALUE DELIMITED BY SIZE
    INTO STRING-BUFFER.
CALL "NewStringPlatform"
    USING BY VALUE JNIEnvPtr
        ADDRESS OF STRING-BUFFER
        ADDRESS OF parmString
        0
    RETURNING RC
IF RC NOT = ZERO THEN
    DISPLAY "Could not create jstring Object"
    GOBACK.
* Call the Java code to generate PDF
    INVOKE PdfCreator "createStatement"
    USING BY VALUE parmString.

GOBACK.

```

4. Paste the COBOL code into the editor and save it (Figure 3-49).

Line 1	Column 1	Insert
1	1	*****
2	1	* COBLOAN *
3	1	* *
4	1	* A simple program that calculates payment amount for a *
5	1	* loan, stores the calculation into a DB2 table, and *
6	1	* generates a PDF file containing the calculation. *
7	1	* *
8	1	* Example input: '30000 .09 24 ' *
9	1	* *
10	1	*****
11	1	IDENTIFICATION DIVISION.
12	1	PROGRAM-ID. 'COBLOAN' RECURSIVE.
13	1	ENVIRONMENT DIVISION.
14	1	CONFIGURATION SECTION.
15	1	REPOSITORY.
16	1	Class PdfCreator is "PdfCreator"
17	1	Class jstring is "jstring".
18	1	DATA DIVISION.
19	1	WORKING-STORAGE SECTION.
20	1	01 FIELDS.
21	1	05 PAYMENT PIC S9(9)V99 USAGE COMP.
22	1	05 PAYMENT-OUT PIC \$\$\$,\$\$\$,\$\$9.99 USAGE DISPLAY.
23	1	05 LOAN-AMOUNT PIC S9(7)V99 USAGE COMP.
24	1	05 LOAN-AMOUNT-IN PIC X(16).

Figure 3-49 Editor with sample COBOL code

3.4.2 Generating the JCL to compile, link, bind, and run COBLOAN

RDz can be used to generate JCL files with some customization to the RDz PROCs. This allows you to specify information such as the data sets and compiler options. Each site has its own configurations.

Example 3-6 provides the compile, link, and bind JCL that we used in our scenario.

Example 3-6 Sample compile and link JCL

```
//COMPLINK JOB (ITSO),'ZAID FAYDI',CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
/*JOBPARM L=9999,SYSAFF=*
//DELETE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE FAYDI.ITSO.LOAD(COBLOAN) PURGE
DELETE FAYDI.ITSO.OBJLIB(COBLOAN) PURGE
DELETE FAYDI.ITSO.DBRLM(COBLOAN) PURGE

DELETE FAYDI.ITSO.LISTING(COBLOAN) PURGE
SET MAXCC = 0
/*
//COMPILE EXEC PGM=IGYCRCTL,
// PARM=(RENT,EXPORTALL,SQL,DLL,LIB,THREAD,PGMNAME(LONGMIXED),
// TEST,DYNAM,QUOTE,NOWORD)
//STEPLIB DD DSN=IGY.SIGYCOMP,DISP=SHR
// DD DSN=DB9J9.SDSNLOAD,DISP=SHR
//SYSLIB DD DSN=FAYDI.ITSO.COPYLIB(JNI),DISP=SHR
//SYSLIN DD DSN=FAYDI.ITSO.OBJLIB(COBLOAN),DISP=SHR
//DBRMLIB DD DSN=FAYDI.ITSO.DBRLM(COBLOAN),DISP=SHR
//SYSPRINT DD DSN=FAYDI.ITSO.LISTING(COBLOAN),DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN DD DSN=FAYDI.ITSO.COBOL(COBLOAN),DISP=SHR
//LKED EXEC PGM=IEWL,
// PARM='RENT,LIST,XREF,LET,MAP,DYNAM(DLL),CASE(MIXED)'
//SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR
// DD DSN=DB9J9.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//SYSLMOD DD DSN=FAYDI.ITSO.LOAD(COBLOAN),DISP=SHR
//SYSDEFSD DD DUMMY
//OBJMOD DD DSN=FAYDI.ITSO.OBJLIB(COBLOAN),DISP=SHR
//SYSLIN DD *
INCLUDE OBJMOD
INCLUDE SYSLIB(DSNRLI)
INCLUDE '/usr/lpp/cobol/lib/igzcjava.x'
INCLUDE '/usr/lpp/java/J6.0.1/bin/j9vm/libjvm.x'
/*
//BIND EXEC PGM=IKJEFT01,COND=(4,LT)
//STEPLIB DD DSN=IGY.SIGYCOMP,DISP=SHR
// DD DSN=DB9J9.SDSNLOAD,DISP=SHR
// DD DSN=DB9J9.SDSNEXIT,DISP=SHR
//DBRMLIB DD DSN=FAYDI.ITSO.DBRLM(COBLOAN),DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD DUMMY
```

```

//SYSUDUMP DD DUMMY
//SYSOUT DD SYSOUT=*
//SYSTSIN DD *
  DSN SYSTEM(DB9J)
    BIND PACKAGE(NULLID) -
      MEMBER(COBLOAN) -
      ISO(CS) -
      ENCODING(EBCDIC) -
      OWNER(FAYDI) -
      ACTION(REPLACE) -
      LIBRARY('FAYDI.ITSO.DBRLM')
  END
/*

```

Example 3-7 provides the JCL we used to run the z/OS Batch Runtime, and add the members COMPLINK and BCDRUN, using the steps described in “Adding COBLOAN” on page 41.

Example 3-7 JCL to run z/OS Batch Runtime

```

//BCDRUN JOB (ITSO),'Zaid',CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
/*JOBPARM L=9999,SYSAFF=*
/*
//PROCLIB JCLLIB ORDER=FAYDI.PROCLIB
//BATCH EXEC BCDPROC,REGION=OM,LOGLVL='+I'
/*
//STEPLIB DD DISP=SHR,DSN=FAYDI.JZOS.LAUNCHER
// DD DISP=SHR,DSN=FAYDI.ITSO.LOAD
// DD DISP=SHR,DSN=DB9J9.SDSNLOAD
// DD DISP=SHR,DSN=DB9J9.SDSNLOAD2
/*
//STDENV DD *
#
#-----
# UPDATE: Installation path for batch runtime.
#-----
export BCD_HOME=/usr/lpp/bcp
#
#-----
# UPDATE: Installation path for Java.
#-----
export JAVA_HOME=/usr/lpp/java/J6.0.1
#
#-----
# The following runs the batch runtime configuration script.
# This script processes the exported environment variables that
# were defined above.
#-----
. $BCD_HOME/bcdconfig.sh
#
#-----
# UPDATE: Uncomment and update JDBC driver files.
#-----
JDBC_HOME="/usr/lpp/db2/db9j/db2910_jdbc"
CLASSPATH="$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"
#
#-----
#   UPDATE: Add your application jar files to the CLASSPATH here.
#-----
CPATH="/u/faydi/com/sample"
CLASSPATH="$CLASSPATH":$CPATH/src/
CLASSPATH="$CLASSPATH":$CPATH/lib/itextpdf-5.1.2.jar
export CLASSPATH="$CLASSPATH"
#
#-----
#   UPDATE: Uncomment and add any additional JVM options here.
#-----
IJO="-Xms256m -Xmx512m -Xquickstart"
#-----
#   UPDATE: Uncomment to enable batch runtime tracing
#-----
IJO="$IJO -Dcom.ibm.zos.batch.container.BCDTraceConfig.trace=all"
#
#-----
#   UPDATE: Add any JDBC options here.  See the DB2 Information
#   Center for details on the options.
#-----
IJO="$IJO -Ddb2.jcc.ssid=DB9J"
#
#-----
#   Exports JVM options set above.
#-----
export IBM_JAVA_OPTIONS="$IJO "
#
#-----
#   The following runs the batch runtime configuration completion
#   script.  This command must be last in the STDENV file.
#-----
. $BCD_HOME/bcdconfigend.sh
/*
/* Batch Runtime Options
/*
/*BCDIN DD *
#-----*
# The following sets the language and class name for the IVP program.
#-----*
bcd.applicationLanguage=COBOL
bcd.applicationName=COBLOAN
bcd.applicationArgs.1=30000 .09 24
#
#-----*
# UPDATE: Support class name used to manage transactions.
#
#       For the DB2 JDBC driver, uncomment the following statement.
#-----*
bcd.supportClass.1=com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport

```

```
#
#-----*
# UPDATE: Verbose mode for additional diagnostics (default is false).
#-----*
bcd.verbose=true
```

3.4.3 Creating the Java launcher and copying the JNI copybook

The JNI copybook contains sample COBOL data definitions that correspond to the Java JNI types. It also contains the JNINativeInterface and the JNI container structure that contains function pointers for accessing the JNI callable services.

The JNI.cpy file is located in the subdirectory of the COBOL install directory, typically /usr/lpp/cobol/include.

- ▶ You can use RDz to copy an HFS file to a MVS partitioned data set (Figure 3-50).

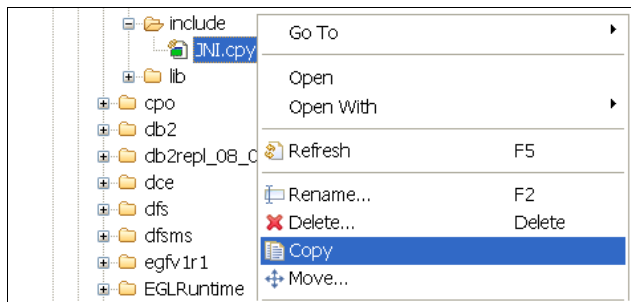


Figure 3-50 Copy of HFS file

- ▶ Then paste it to MVS partitioned data set, as shown in Figure 3-51.

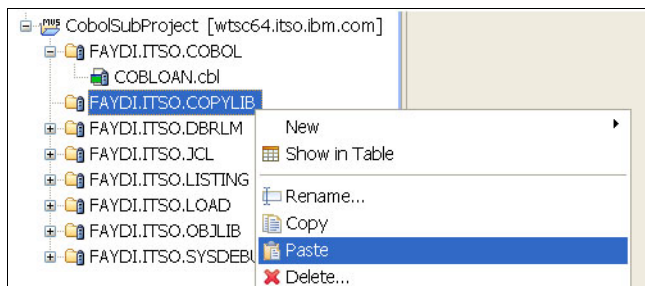


Figure 3-51 Paste to MVS data set

The JZOS Launcher for V6.0.1(JVMLDM61) is located in the mvstools subdirectory of the JVM install, typically /usr/lpp/java/J6.0.1/mvstools/.

Drag and drop the iText library to the lib folder

The Java archive file itextpdf-5.1.2.jar should be copied to your lib folder under the sample filter in JavaSubProject. iText is a Java library that allows you to create and manipulate PDF documents. This library will be used to create statements containing the calculations made for customers to view.

1. In the Remote Systems view, expand the Local node.
2. Expand the My Home filter and navigate to the location where the archive file is stored.
3. After it is located, drag and drop the file straight to the lib folder.

3.4.4 Creating the PdfCreator class

PdfCreator is a Java class that extends the current COBOL functionality to create a PDF document containing the calculations made in COBLOAN. It also updates the record created in COBLOAN containing those calculations with the location the PDF document.

1. To create an HFS file, right-click the **src** folder under **JavaSubProject New** → **HFS File** (Figure 3-52).

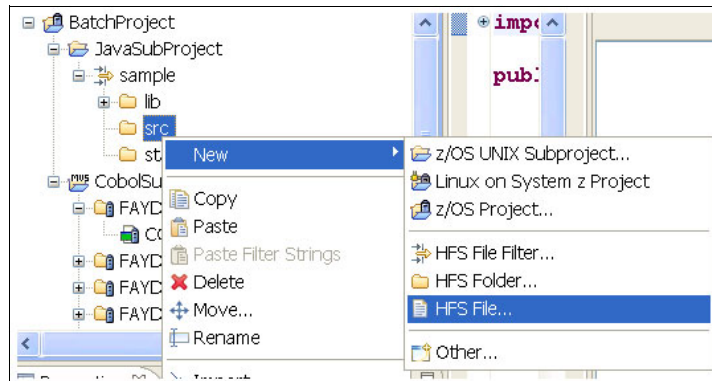


Figure 3-52 Create new HFS file

2. Enter the file name and click **Finish** (Figure 3-53).

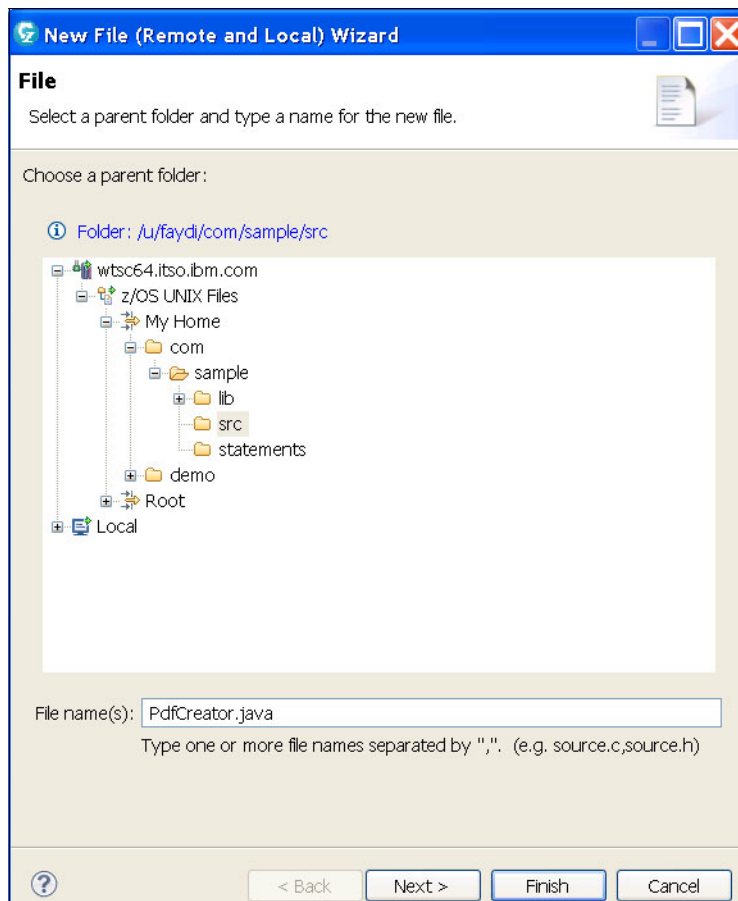


Figure 3-53 Enter name for new file

3. Double-click the newly created file to open it in the editor, then copy into it the Java source code from Example 3-8.

Example 3-8 PdfCreator.java

```
import java.io.*;
import com.itextpdf.text.*;
import com.itextpdf.text.pdf.*;
import com.itextpdf.text.pdf.draw.*;
import java.sql.*;
import com.ibm.batch.spi.UserControlledTransactionHelper;
import java.util.StringTokenizer;

import java.text.SimpleDateFormat;
import java.util.Calendar;

public class PdfCreator {

    private static final String STATEMENTS_PATH =
"/u/faydi/com/sample/statements/";
    private static final String URL= "jdbc:default:connection";

    public static void createStatement (String stringValues) {
        try {

            // Parse words from the parameter String
            System.out.println("Parameter: " + stringValues);
            String[] values = stringValues.trim().split(" ");

            System.out.println("Words in parameter: " + values.length);
            if (values.length != 4) {
                throw new Exception("Unexpected number of words in parameter '" +
stringValues + "'");
            }

            // Data type conversions
            String payment = values[0];
            double amount = Double.parseDouble(values[1]);
            double interest = Double.parseDouble(values[2]);
            int periods = Integer.parseInt(values[3]);

            // Get timestamp to uniquely name PDF files
            Calendar calendar = Calendar.getInstance();
            String path = STATEMENTS_PATH + calendar.getTimeInMillis() + ".pdf";
            System.out.println(path);

            FileOutputStream fileOutputStream = new FileOutputStream(path);
            Document document = new Document();
            PdfWriter.getInstance(document, fileOutputStream);
            document.open();
            Chunk chunk;

            // Format and create the PDF file
            Paragraph paragraph = new Paragraph();
            chunk = new Chunk("Amount: " + amount + "\n");
            paragraph.add(new Chunk(chunk));
```

```

        chunk = new Chunk("Interest: " + interest + "\n");
        paragraph.add(new Chunk(chunk));
        chunk = new Chunk("Periods: " + periods + "\n");
        paragraph.add(new Chunk(chunk));
        chunk = new Chunk("Payment: " + payment + "\n");
        paragraph.add(new Chunk(chunk));
        document.add(paragraph);

        document.close();
        System.out.println("PDF generated.");

        // Update QUERIES table with path
        Connection connection = DriverManager.getConnection(URL);
        Statement statement = connection.createStatement();
        String sql = "UPDATE QUERIES SET PATH='" + path + "' WHERE " +
            "AMOUNT=" + amount + " AND INTEREST=" + interest + " AND " +
            "PERIODS=" + periods;

        System.out.println("About to execute SQL: " + sql);
        statement.executeUpdate(sql);
        UserControlledTransactionHelper.commit();
        System.out.println("Record update committed");
    }
    catch (Exception exception) {
        System.err.println("An exception has occurred, Rollback changes");
        System.err.println(exception.getMessage());
        // Handling an exception that occurred to rollback
        try {
            UserControlledTransactionHelper.rollback();
        }
        catch (Exception exception2) {
            System.err.println("Rollback failed");
            System.err.println(exception2.getMessage());
        }
        System.err.println("Rollback complete");
    }
}
}
}

```

3.4.5 Compiling PdfCreator

The following steps demonstrate one way to compile the PdfCreator.

1. Copy the JCL shown in Example 3-9.

Example 3-9 javac command to compile PdfCreator

```

//JAVAC JOB (ITS0),'ZAID FAYDI',CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),
// REGION=0M
/*JOBPARM L=9999,SYSAFF=*
//RUNSHELL EXEC PGM=BPXBATCH,
// PARM='sh javac -verbose /u/faydi/com/sample/src/PdfCreator.java'
//STDOUT DD SYSOUT=*

```

2. Create a new member called JAVAC under <HLQ>.ITS0.JCL and paste the JCL into it.
3. Right-click **JAVAC.jcl** and select **Submit**. Keep refreshing the My Jobs filter until the job is complete (Figure 3-55).

Double-click **RUNSHELL: STDOUT**. You should see output similar to Example 3-10.

Example 3-10 RUNSHELL:STDOUT

```

Ÿparsing started /u/faydi/com/sample/src/PdfCreator.java"
Ÿparsing completed 44ms
Ÿsearch path for source files:
./u/faydi/com/samples/lib/itextpdf-5.1.2.jar,/u/faydi/demo/samples/lib/itextpdf-5
.1.2.jar,/usr/lpp/b
Ÿsearch path for class files:
/Z1DRB1/usr/lpp/java/J6.0.1/lib/s390/default/jc1SC160/vm.jar,/Z1DRB1/usr/lpp/java/
J6.0.1/lib/annotatio
...
Ÿwrote /u/faydi/com/sample/src/PdfCreator.class"
Ÿtotal 1289ms"

```

4. The PdfCreator.class should appear under the src folder (Figure 3-54).

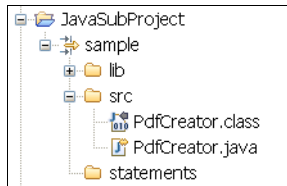


Figure 3-54 PdfCreator.class in src folder

5. The JAVAC job will appear under the My Jobs filter under the JES node in the Remote Systems view (Figure 3-55). Refresh this filter until the job completes.

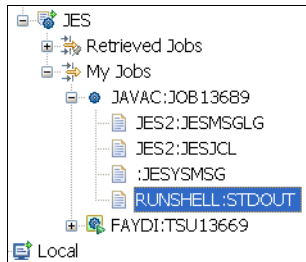


Figure 3-55 JAVAC job

3.4.6 Running COBLOAN in the container

Use the following steps to run COBLOAN in the container.

1. The JCL to run COBLOAN in the container was created in 3.4.2, “Generating the JCL to compile, link, bind, and run COBLOAN” on page 44.

Submit BCDRUN and refresh the My Jobs filter until it is complete (Figure 3-56).

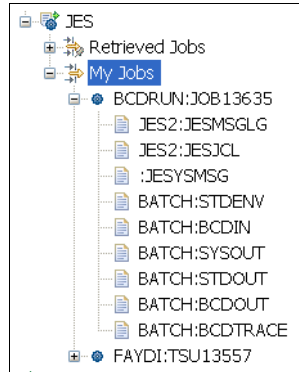


Figure 3-56 My Jobs filters

2. As outlined in “Viewing application and container output” on page 8, all output from the JVM (messages prefixed with JVM) and COBLOAN (no message codes) appears in SYSOUT. The DISPLAY calls from COBLOAN are highlighted in Example 3-11. The output shows that the calculations record was successfully inserted into the QUERIES table.

Example 3-11 SYSOUT

```
JVMJZBL2004N Log level has been set to: I
JVMJZBL1001N JZOS batch Launcher Version: 2.3.0 2010-12-14
JVMJZBL1002N Copyright (C) IBM Corp. 2005. All rights reserved.
JVMJZBL1029I Region requested = 0K, Actual below/above limit = 8168K / 1461M
JVMJZBL1053I OS Release R23.00 Machine 2094
JVMJZBL1005I Output from DD:STDENV config shell script:
...
JVMJZBL1023N Invoking com.ibm.zos.batch.container.BCDBatchContainer.main()...
Payment          $1,370.54
30000          .09 24          $1,370.54
insert record successful
JVMJZBL1024N com.ibm.zos.batch.container.BCDBatchContainer.main() completed.
JVMJZBL1014I Waiting for non-deamon Java threads to finish before exiting...
JVMJZBL2999I JZOS batch launcher elapsed time=5.869020 seconds, cpu time=3.3427240
seconds
JVMJZBL1021N JZOS batch launcher completed, return code=0
```

- To check that the record was inserted, open the Data perspective, navigate to the QUERIES table in the Data Source Explorer view, right-click it, and select **Data** → **Return All Rows** (Figure 3-57).

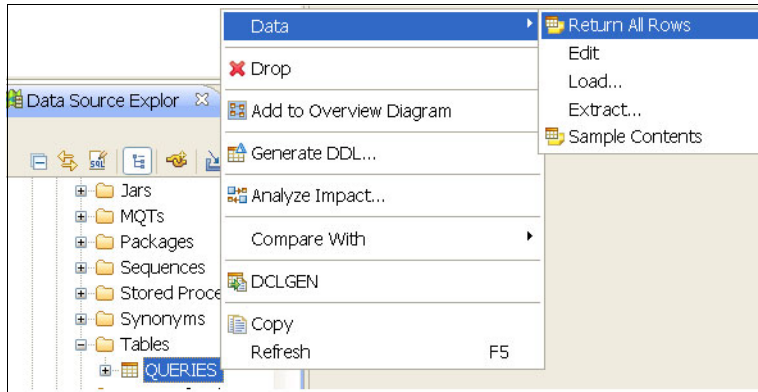


Figure 3-57 QUERIES table in Data Source Explorer

- The output for the query is shown in Figure 3-58.

Status	Result1	AMOUNT	INTEREST	PERIODS	PAYMENT	PATH
1		30000.00	0.09	24	\$1,370.54	/u/faydi/com/sample/statements/1320734128585.pdf

Figure 3-58 QUERIES output

- The System.out output from PdfCreator appears in STDOUT. From the output (Example 3-12) it appears that the PDF file was created and the QUERIES record (inserted in COBLOAN) was updated to include the path of the PDF file.

Example 3-12 STDOUT

```

Parameter:          $1,370.54 30000 .09 24
Words in parameter: 4
/u/faydi/com/sample/statements/1320734128585.pdf
PDF generated.
About to execute SQL: UPDATE QUERIES SET
PATH='/u/faydi/com/sample/statements/1320734128585.pdf' WHERE AMOUNT=30000.0 AND
INTEREST=0.09 AND PERIODS=24
Record update committed
  
```

- To view the PDF file, open and refresh the statements folder. Right-click the PDF document file and select **Open With** → **System Editor** (Figure 3-59).

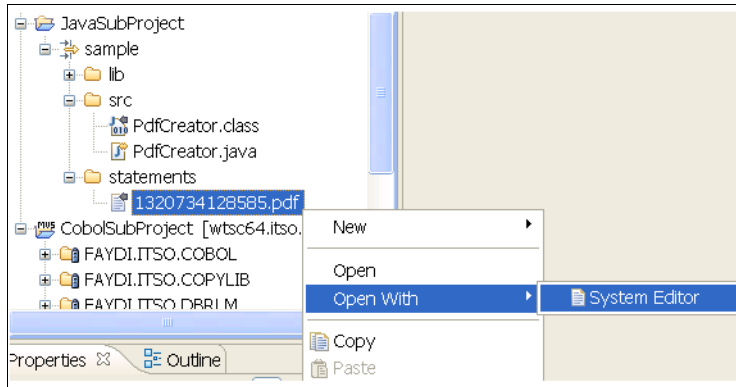


Figure 3-59 Open pdf with System Editor

- The output is shown in Figure 3-60.

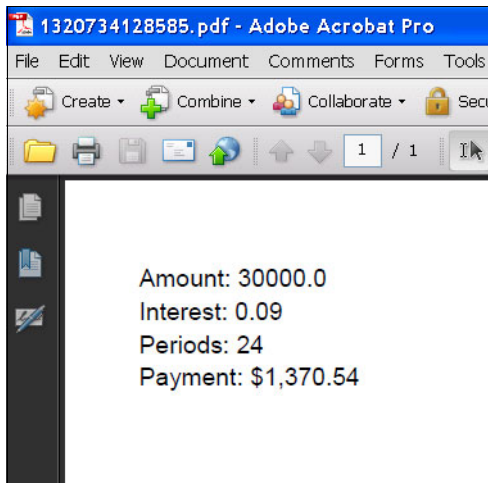


Figure 3-60 View of output

The BCDOUT outputs z/OS Batch Runtime messages. The BCD0412I message is key because it counts the number of transactions that were issued, committed, and rolled back.

Example 3-13 shows that two transactions were committed successfully:

- ▶ INSERT SQL statement in COBLOAN
- ▶ UPDATE SQL statement in PdfCreator

Example 3-13 BCDOUT

```
BCD0206I Batch Runtime started at Tue Nov 08 01:35:25 EST 2011 (build
2011011_130751701_mt19483 framework BATCC10.BATCH Ycf011104.03
..).
BCD0218I Batch Runtime options in effect:
BCD0219I   bcd.applicationArgs.1=30000 .09 24
BCD0219I   bcd.applicationLanguage=COBOL
BCD0219I   bcd.applicationName=COBLOAN
BCD0219I   bcd.supportClass.1=com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport
BCD0219I   bcd.verbose=true
```

```

BCD0230I Class "com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport" was loaded from
file:/pp/db2v9/D110202/db2910_jdbc/classes/db2jcc.
jar.
BCD0208I Initialization started for Batch Runtime support class
"com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport".
BCD0227I Batch Runtime support class
"com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport" version information: IBM DB2
JDBC Universal
Driver Architecture 3.61.84.
BCD0209I Initialization complete for Batch Runtime support class
"com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport".
BCD0406I Begin transaction processing started at Tue Nov 08 01:35:28 EST 2011.
BCD0407I Begin transaction processing completed at Tue Nov 08 01:35:28 EST 2011.
BCD0303I Launching application "COBLOAN".
BCD0408I Commit transaction processing started at Tue Nov 08 01:35:29 EST 2011.
BCD0409I Commit transaction processing completed at Tue Nov 08 01:35:29 EST 2011.
BCD0406I Begin transaction processing started at Tue Nov 08 01:35:29 EST 2011.
BCD0407I Begin transaction processing completed at Tue Nov 08 01:35:29 EST 2011.
BCD0305I Application "COBLOAN" completed: return code=0.
BCD0408I Commit transaction processing started at Tue Nov 08 01:35:29 EST 2011.
BCD0409I Commit transaction processing completed at Tue Nov 08 01:35:29 EST 2011.
BCD0214I Termination started for Batch Runtime support class
"com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport".
BCD0215I Termination complete for Batch Runtime support class
"com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport".
BCD0412I Transaction counts: Begin=2 Commit=2 Rollback=0.
BCD0225I Batch Runtime ended at Tue Nov 08 01:35:29 EST 2011.

```

Demonstrating a rollback

The COBLOAN INSERT and PdfCreator UPDATE database changes are defined as a single transaction by the `UserControlledTransactionHelper.commit()` call at the end of PdfCreator. PdfCreator is only called when the COBLOAN INSERT is successful. If any exceptions occur in PdfCreator, then the INSERT from COBLOAN is discarded. One possible cause of an exception in PdfCreator is if the statements folder cannot be found.

We took the following steps to demonstrate a rollback.

1. Select the Statements folder and click **Rename** (Figure 3-61).

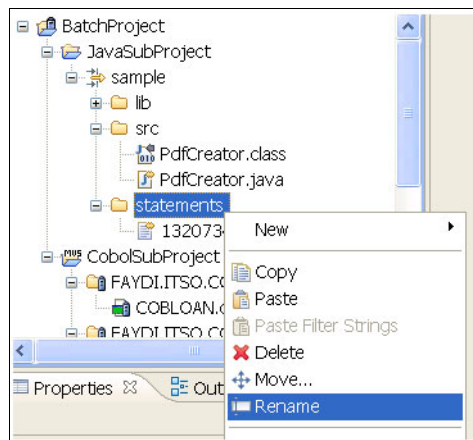


Figure 3-61 Rename statements folder

2. In the Name dialog, enter pdfs as the name of the folder (Figure 3-62).

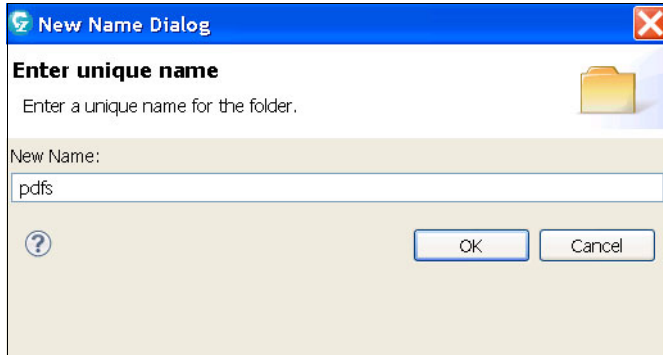


Figure 3-62 New Name Dialog: Enter unique name

3. The pdfs folder is created (Figure 3-63).

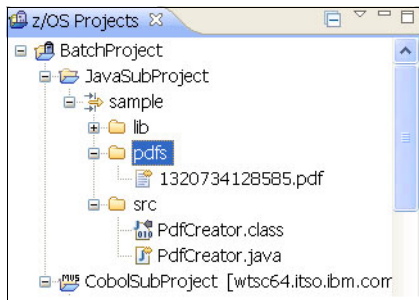


Figure 3-63 pdfs folder

► Example 3-14 demonstrates how this rollback changes the input values in BCDIN.

Example 3-14 Changing COBLOAN arguments

```

...
/**
/** Batch Runtime Options
/**
//BCDIN DD *
#-----*
# The following sets the language and class name for the IVP program.
#-----*
bcd.applicationLanguage=COBOL
bcd.applicationName=COBLOAN
bcd.applicationArgs.1=125000 .07 36
#
...

```

4. Submit the BCDRUN JCL. See Example 3-15 for the output in STDOUT. The STDOUT output from this execution also suggests that it did not complete compared to the STDOUT output from the previous execution shown in see Example 3-12 on page 54.

Example 3-15 STDOUT

```

Parameter:      $3,859.64 125000 .07 36
Words in parameter: 4
/u/faydi/com/sample/statements/1320739595362.pdf

```

5. The output for STDERR is shown in Example 3-16. The output in STDERR is due to the System.err calls in the exception handling catch block of PdfCreator and indicates that an exception occurred.

Example 3-16 STDERR

```
An exception has occurred, Rollback changes
/u/faydi/com/sample/statements/1320739595362.pdf (EDC5129I No such file or
directory.)
Rollback complete
```

6. The z/OS Batch Runtime output in BCDOUT shows that one change was committed and then rolled back (Example 3-17).

Example 3-17 BCDOUT

```
BCD0206I Batch Runtime started at Tue Nov 08 03:06:33 EST 2011 (build
2011011_130751701_mt19483 framework BATCC10.BATCH Ycf011104.03
").
BCD0218I Batch Runtime options in effect:
BCD0219I   bcd.applicationArgs.1=125000 .07 36
BCD0219I   bcd.applicationLanguage=COBOL
BCD0219I   bcd.applicationName=COBLOAN
BCD0219I   bcd.supportClass.1=com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport
BCD0219I   bcd.verbose=true
BCD0230I Class "com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport" was loaded from
file:/pp/db2v9/D110202/db2910_jdbc/classes/db2jcc.
jar.
BCD0208I Initialization started for Batch Runtime support class
"com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport".
BCD0227I Batch Runtime support class
"com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport" version information: IBM DB2
JDBC Universal
Driver Architecture 3.61.84.
BCD0209I Initialization complete for Batch Runtime support class
"com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport".
BCD0406I Begin transaction processing started   at Tue Nov 08 03:06:35 EST 2011.
BCD0407I Begin transaction processing completed at Tue Nov 08 03:06:35 EST 2011.
BCD0303I Launching application "COBLOAN".
BCD0410I Rollback transaction processing started   at Tue Nov 08 03:06:35 EST
2011.
BCD0411I Rollback transaction processing completed at Tue Nov 08 03:06:35 EST
2011.
BCD0406I Begin transaction processing started   at Tue Nov 08 03:06:35 EST 2011.
BCD0407I Begin transaction processing completed at Tue Nov 08 03:06:35 EST 2011.
BCD0305I Application "COBLOAN" completed: return code=0.
BCD0408I Commit transaction processing started   at Tue Nov 08 03:06:35 EST 2011.
BCD0409I Commit transaction processing completed at Tue Nov 08 03:06:35 EST 2011.
BCD0214I Termination started for Batch Runtime support class
"com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport".
BCD0215I Termination complete for Batch Runtime support class
"com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport".
BCD0412I Transaction counts: Begin=2 Commit=1 Rollback=1.
BCD0225I Batch Runtime ended at Tue Nov 08 03:06:35 EST 2011.
```

7. No additional PDF document files were created (Figure 3-64).

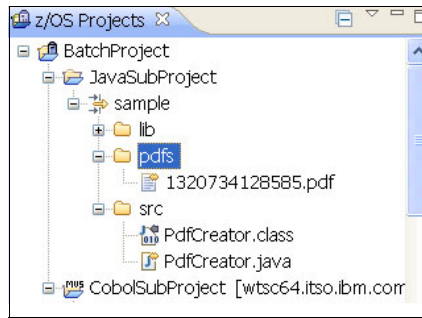


Figure 3-64 View of pdfs folder

8. There are no additional records in the QUERIES table (Figure 3-65).

Status	Result1	AMOUNT	INTEREST	PERIODS	PAYMENT	PATH
1		30000.00	0.09	24	\$1,370.54	/u/faydi/com/sample/statements/1320734128585.pdf

Figure 3-65 View of QUERIES table

9. SYSOUT in the z/OS Batch Runtime returned a return code of 0 (Example 3-18). However, this is not a true reflection of the execution of COBLOAN. A proposed solution is outlined in the next section, 3.4.7, “Solution to reflect an exception in the return code”.

Example 3-18 z/OS Batch Runtime return code 0

```

...
VMJZBL1023N Invoking com.ibm.zos.batch.container.BCDBatchContainer.main()...
Payment          $3,859.64
125000           .07  36      $3,859.64
insert record successful
JVMJZBL1024N com.ibm.zos.batch.container.BCDBatchContainer.main() completed.
JVMJZBL1014I Waiting for non-deamon Java threads to finish before exiting...
JVMJZBL2999I JZOS batch launcher elapsed time=4.973982 seconds, cpu time=2.2908413
seconds
JVMJZBL1021N JZOS batch launcher completed, return code=0

```

3.4.7 Solution to reflect an exception in the return code

The current version of PdfCreator does not have any logic in the catch block to return an integer value representing the return code, nor is there any logic in COBLOAN to handle return integer value from PdfCreator. Refer to 2.2.4, “Handling return codes and Java exceptions” on page 9. The following is a way to implement passing back a return code from the application and its components.

1. In the PdfCreator, add a return type to the createStatement method (Example 3-19).

Example 3-19 Changing return type from void to int

```

...
public static int createStatement (String stringValues) {
...

```

2. Add the return statements to return a 0 or 12 (Example 3-20).

Example 3-20 Adding return statements to return a return code

```
...
    catch (Exception exception) {
        System.err.println("An exception has occurred, Rollback changes");
        System.err.println(exception.getMessage());
        // Handling an exception that occurred to rollback
        try {
            UserControlledTransactionHelper.rollback();
        }
        catch (Exception exception2) {
            System.err.println("Rollback failed");
            System.err.println(exception2.getMessage());
            return 12;
        }
        System.err.println("Rollback complete");
        return 12;
    }
return 0;
```

3. Modify COBLOAN to accept a return code from PdfCreator and MOVE that value to the RETURN-CODE register (Example 3-21).

Example 3-21 Adding return code handling to COBLOAN

```
...
    * Call the Java code to generate PDF
      INVOKE PdfCreator "createStatement"
      USING BY VALUE parmString
      RETURNING RC.
      IF RC > RETURN-CODE THEN
        MOVE RC TO RETURN-CODE.

      GOBACK.
```

4. Recompile COBLOAN via JCL COMPLINK and PdfCreator via JCL JAVAC. Submit BCDRUN when all the applications have been recompiled successfully. You should see a return code of 12 as shown in Example 3-22.

Example 3-22 BCDOUT

```
...
BCD0303I Launching application "COBLOAN".
BCD0410I Rollback transaction processing started   at Tue Nov 08 04:00:35 EST
2011.
BCD0411I Rollback transaction processing completed at Tue Nov 08 04:00:35 EST
2011.
BCD0406I Begin transaction processing started   at Tue Nov 08 04:00:35 EST 2011.
BCD0407I Begin transaction processing completed at Tue Nov 08 04:00:35 EST 2011.
BCD0305I Application "COBLOAN" completed: return code=12.
BCD0408I Commit transaction processing started   at Tue Nov 08 04:00:35 EST 2011.
BCD0409I Commit transaction processing completed at Tue Nov 08 04:00:35 EST 2011.
BCD0214I Termination started for Batch Runtime support class
"com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport".
```

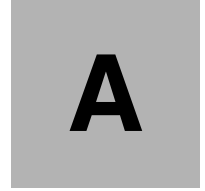
BCD0215I Termination complete for Batch Runtime support class
"com.ibm.db2.jcc.t2zos.T2zosBatchContainerSupport".
BCD0412I Transaction counts: Begin=2 Commit=1 Rollback=1.
BCD0225I Batch Runtime ended at Tue Nov 08 04:00:35 EST 2011.

5. The JCL for the z/OS Batch Runtime BATCH step returns a return code of 8 (Example 3-23). As described in "Handling return codes and Java exceptions" on page 9 it indicates that the primary application, COBLOAN, returned a return code greater than 0.

Example 3-23 JES JOB LOG

-JOBNAME	STEPNAME	PROCSTEP	RC	EXCP	CPU
-BCDRUN	BATCH	JAVA	08	37900	.06

To reproduce this development scenario without having to copy/paste the source code listed in this chapter, you can be download the necessary files from the Internet as described in Appendix A., "Additional material" on page 63.



Additional material

This book refers to additional material that can be downloaded from the Internet as described in the following sections.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks web server. Point your web browser at:

<ftp://www.redbooks.ibm.com/redbooks/SG248116>

Alternatively, you can go to the IBM Redbooks website at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG248116.

Using the Web material

The additional Web material that accompanies this book includes the following files:

<i>File name</i>	<i>Description</i>
zOS_Batch_Runtime_sample.zip	z/OS Batch Runtime development scenario as described in “End-to-end development scenario” on page 17.

System requirements for downloading the Web material

The Web material requires the following system configuration:

Hard disk space: 1 MB minimum
Operating System: Windows

Downloading and extracting the Web material

Create a subdirectory (folder) on your workstation, and extract the contents of the Web material .zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks publications

The following IBM Redbooks publication provides additional information about the topic in this document. Note that some publications referenced might be available in softcopy only.

- ▶ *Batch Modernization on z/OS*, SG24-7779

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Other publications

These publications are also relevant as further information sources:

- ▶ *z/OS Batch Runtime: Planning and User's Guide*, SA23-7270
- ▶ *IBM DB2 Universal Database Application Development Guide: Programming Client Applications*, SC09-4826

Online resources

- ▶ z/OS Java Technology Edition
http://www-03.ibm.com/systems/z/os/zos/tools/java/products/sdk601_31.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services





New Ways of Running IBM z/OS Batch Applications



**Modernizing
enterprise batch**

**Designing hybrid
applications**

**Implementing a
practical example**

Mainframe computers play a central role in the daily operations of many of the world's largest corporations. Batch processing is still a fundamental, mission-critical component of the workloads that run on the mainframe and a large portion of the workload on IBM z/OS systems is processed in batch mode.

This IBM Redbooks publication is the second volume in a series of four in which we describe new technologies introduced by IBM to facilitate the use of hybrid batch applications that combine the best aspects of Java and procedural programming languages such as COBOL. This volume specifically focuses on z/OS batch runtime.

The audience for this book includes IT architects and application developers, with a focus on batch processing on the z/OS platform.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:
ibm.com/redbooks**

SG24-8116-00

ISBN 0738437956