IBM

New Ways of Running Batch Applications on z/OS

Volume 1 CICS Transaction Server

- Technology overview
- Application design considerations
- Example

Daniel Millwood
Alex Louwe Kooijmans
Elsie Ramos

Redbooks

ibm.com/redbooks

International Technical Support Organization

New Ways of Running Batch Applications on z/OS: Volume 1 CICS Transaction Server

October 2013

SG24-7991-00

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

First Edition (October 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, V7
- WebSphere Application Server Network Deployment for z/OS V8.5
- CICS Transaction Server for z/OS V4.2
- Rational Application Developer V8.0.3 iFix1

Contents

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 give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

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.

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 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 Explorer®
CICS®
DB2®
IBM®
IMS™
MVS™
Rational®
Redbooks®
Redbooks (logo) ®
System z®
Tivoli®
WebSphere®
z/OS®

The following terms are trademarks of other companies:

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

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 world's largest corporations. Batch processing is still a fundamental, mission-critical component of the workloads that run on the mainframe. A large portion of the workload on IBM® z/OS® systems is processed in batch mode.

This IBM Redbooks® publication is the first volume in a series in which we specifically address new technologies introduced by IBM to facilitate new ways to run batch applications on z/OS that combine the best aspects of Java and procedural programming languages such as COBOL. This volume specifically focuses on the latest support in IBM CICS® to run batch tasks.

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

Authors

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

Daniel Millwood is a Software Developer working within the IBM CICS Transaction Server for z/OS development team in England. He has worked at IBM for 17 years on messaging and transaction processing products. He holds a degree in Computer Science from Southampton University. His areas of expertise include IBM WebSphere® MQ, WebSphere Application Server, and CICS TS. In his role with CICS TS, Dan has focused on the integration of CICS TS with other products, using technologies such as web services.

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 various job roles in application development. His current focus is on modernizing core banking systems and the role of the current IBM 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 also to the following individuals:

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

Ashley Bill
IBM Developer, IBM Hursley

Mark Cocker
IBM United Kingdom Limited

John Hutchinson
IBM USA

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

Susann Thomas
IBM Germany

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

1

Introduction to CICS batch support for WebSphere Batch Environment

This IBM Redbooks publication discusses the CICS TS Feature Pack for Modern Batch. This capability can be installed into CICS Transaction Server for z/OS (CICS TS) V4.2 or higher and enables the WebSphere Batch Environment to schedule and manage batch applications in CICS.

This book reviews the types of workloads it is suitable for and the architecture of the overall solution. We highlight some of the key design considerations when building a batch application to run in CICS and we provide a walkthrough of the development, deployment, testing and debugging a simple CICS batch application.

1.1 What the Batch Environment is

The Batch Environment is a managed environment for batch applications that are scheduled, process large amounts of information, and may take many hours to complete. The Batch Environment provides a powerful failover model based on checkpoint and restart scenarios. This is essential to efficiently manage, run, and restart batch applications, in particular when batch application resources are shared with online transaction processing.

The Batch Environment has two primary components:

1. The Job Scheduler is responsible for determining when and where to dispatch a job, monitoring the job, and reporting back to its caller.
2. Endpoints (Batch Containers) are where the batch application runs. Jobs are dispatched into an Endpoint from the Job Scheduler. The job runs and, upon completion, the job log and return code is provided back to the Job Scheduler.

CICS is a modern general-purpose transaction processing environment for online applications that start as a result of a request received through a terminal, web service, or message. It typically processes a small amount of information within subseconds. CICS provides the following capabilities:

- Administration, security, and transaction facilities, such as authorization, data integrity, workload management, logging, tracing, debugging, statistics, and monitoring
- API and development tools, such as named counter and XML conversion
- Shared access to resources, such as temporary storage, data tables, IBM DB2®, IBM IMS™, and VSAM
- Communications, such as web services, WebSphere Optimized Local Adapters (WOLA), WebSphere MQ, HTTP, and sockets

In this chapter, we discuss the technology needed to enable an Endpoint to be hosted in CICS, thereby enabling batch applications to run in CICS. These batch applications then have easy access to CICS facilities, such as the resources that CICS manages.

1.2 Conflicting needs of online applications and z/OS batch applications

It is common for an online CICS application to update resources. For some resources, such as VSAM files, CICS maintains a record and image lock to prevent other applications from making conflicting updates and to be able to restore records in case of a failure. In these cases, the resources need to be opened exclusively by CICS.

A traditional z/OS batch application can be defined as a job written in job control language (JCL) that is submitted to the z/OS job entry subsystem (JES) for execution and does not need user input to complete. For example, it reads all records from a VSAM input file and, for each one, updates a VSAM master file and creates a summary report. However, the master file may be opened for exclusive use by CICS and thus be unavailable to the batch application.

In this scenario, there are a number of choices:

- Close the master file in CICS and start the batch application, which starts by taking a backup of the master file.

  Then process all the records, make the updates, delete the backup and re-open the master file in CICS. If the batch application fails, the backup of the master file is restored, and either the issue is fixed immediately and the batch application is re-started, or the issue is fixed at a later time.

  This choice results in a period of time, referred to as the *batch window*, during which the master file and the online applications that use that file are not available.

- Code the batch application to send a request to an online CICS application to make each update to the resources locked by the online application.

  If each request is committed individually, for example, using the CICS non-transactional External Communications Interface (EXCI), this will cause data integrity issues if the batch application fails, because if the batch application is re-started, some updates will be executed twice. If all requests were committed together, for example, using transactional EXCI, this can result in many records being locked by the online application for an extended time, causing unacceptable delays for other applications.

- For VSAM resources, use Record Level Sharing (RLS) to maintain record locks for the batch application. However, the batch application is unlikely to maintain its own recoverable logs due to the complexity of writing them. Therefore, if the batch application fails, the records already updated will not be restored, leading to data integrity issues.

- For VSAM resources, use DFSMS Transactional VSAM services to lock and log record images before updates. However, unless the application implements its own checkpointing, many records can be locked for an extended period, causing unacceptable delays for other applications.

In addition, as companies provide their services across more geographies and time zones, and customers require services at times of day to suit them, there is a growing need for online applications to be available continuously, 24x7. Also over time, batch applications are expected to process an increasing amount of data and there is a need to drive down costs. Therefore, in the event of the batch application failing, it is unacceptable to restart it from the beginning. Instead there is a requirement to restart from a frequent checkpoint.

2

Running batch applications in CICS

In this chapter, we review a solution to resolve the conflicting needs of batch and online applications. It can be used to develop a batch application that uses the batch programming model and runs the application in CICS.

These are the key behavioral aspects of such a batch application:

- The batch application shares access to resources with online applications.
- The batch application takes regular checkpoints to free up transactional resources, so that online applications are not blocked from completing for excessive amounts of time.

  Note: The Endpoint provides this checkpointing capability on behalf of the applications.

- If the batch application fails, it can be restarted from its most recent checkpoint.
- Both batch and online applications run concurrently.

The batch application can be divided into job steps that execute in parallel against different subsets of the input data, to shorten the overall elapsed time to process the job.

2.1 WebSphere Batch Environment architecture

The CICS TS Feature Pack for Modern Batch provides an Endpoint called the Batch Container that runs in a Java Virtual Machine (JVM) in the CICS address space. The Job scheduler interacts with the Batch Container to start, stop, and manage batch applications. Figure 2-1 depicts this architecture and the interaction between the components.

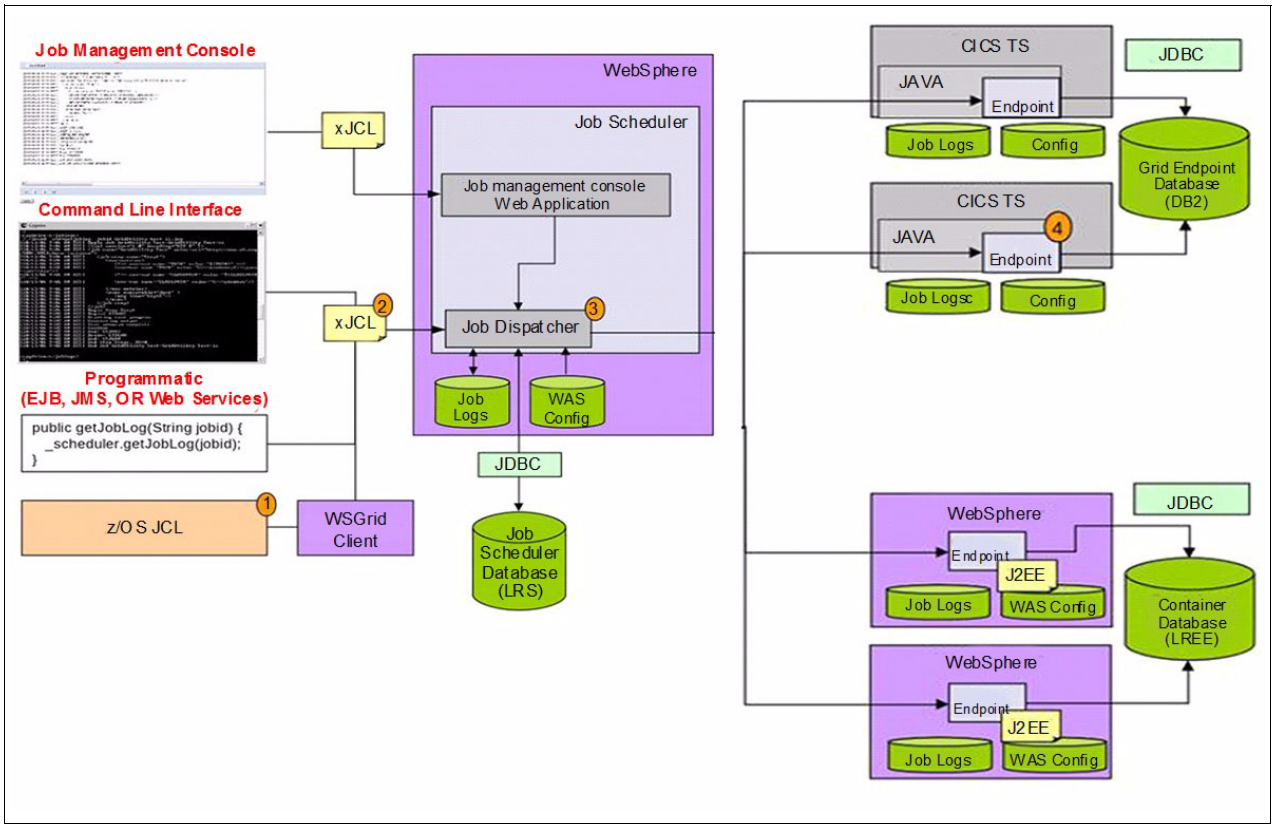

Figure 2-1 CICS provides an Endpoint for the WebSphere Batch Environment

When started in CICS, the Batch Container loads a configuration file that details which batch applications it can run, how to connect to the job scheduler, and how the job scheduler can connect to it. The Batch Container registers with the job scheduler and informs the scheduler of the batch applications it can run. It then periodically sends status information to the scheduler, which states that it is still active and available for work.

The starting and processing of a job are detailed using the numbers in the diagram:

1. JCL is submitted on z/OS to request that the batch application start. The JCL runs program WSGRID and passes it the location of an xJCL document.
2. The WSGRID program connects to the job scheduler and passes the xJCL location. Alternate interfaces are provided to start a batch application, including a console, command line interface, and programmatic API.
3. The job scheduler examines the xJCL to establish the name of the batch application to dispatch and uses the information published to it from all Endpoints to select which Endpoint should run the batch application. The job scheduler chooses a Batch Container hosted in CICS. It then connects to the Batch Container passing it the xJCL.
4. The batch application runs inside the CICS Batch Container.

When the batch application is running in the Batch Container in CICS, it can use Java APIs such as JDBC, or the CICS Java APIs (JCICS), to access CICS resources and services including VSAM files, and to call existing CICS programs written in other languages, such as COBOL, C/C++, PL/I, and Assembler. As the job runs, the Batch Container takes checkpoints, which enables the batch application to be restarted from the last successful checkpoint in the event of a failure. When the batch application completes, the Batch Container notifies the job scheduler.

2.2 When it makes sense to run a batch application in CICS

There are various reasons to run batch applications in CICS:

- When there is pressure to reduce or eliminate the batch window:

  If CICS is used for online transaction processing and those online applications need to be available for longer each day, then it can make sense to run batch jobs in CICS.

- When CICS has opened resources exclusively that are needed for batch:

  If your batch needs access to resources that are opened exclusively by CICS, then it can make sense to run the batch application under the control of CICS.

- When the batch application does not need exclusive resource access:

  If the batch application runs at the same time as online applications, then updates to the resources being used by the batch job can be made by the online applications. To work in this environment, the batch application needs to be tolerant of these changes.

- When you want to re-use CICS business logic in batch:

  Re-use of existing business logic between online applications and batch helps to reduce duplication and can make it quicker and easier to develop new applications. It is likely that there is significant business logic contained within existing CICS online applications. This logic can be invoked using the JCICS equivalent of the EXEC CICS LINK API.

2.3 Benefits of running a batch job inside CICS

The following benefits can result from running batch jobs in CICS:

- It enables online CICS applications to be available closer to 24 hours a day:

  By running online applications and batch applications in parallel, there is less need to take the CICS managed resources offline.

- Capabilities provided by the Batch Container simplify application development:

  Capabilities such as checkpointing, recovery to last checkpoint after a failure, logging, and trace are provided by the Batch Container, removing the need for the application developer to provide these capabilities.

- A common batch programming model is used:

  The Batch Environment provides a common batch programming model across runtimes and platforms. Therefore, any developers skilled in Batch Environment application development should be able to write a batch application to run in CICS.

- Java skills are readily available:

  Java is a well known, popular, modern language. Batch Job Steps and Batch Data Streams are written in Java.

- Java processing is eligible for offloading:

  Java processing can be offloaded to System z Application Assist Processors (zAAPs). This can free up CPU cycles on the main processors and can provide a financial benefit as the charging model for zAAP is not based on the amount of CPU used.

- Java functionality is available off the shelf:

  Functions such as email, PDF generation and XML processing are readily available for Java. This can make it easy to develop batch functions that might be more challenging to implement in languages such as COBOL or PL/I.

2.4 Implications of running a batch job inside CICS

Running a batch job under the control of CICS means that the job has different behavioral characteristics to be aware of. In this section, we highlight some of the implications of running a batch job in CICS.

2.4.1 Batch jobs may take longer to run

Traditional z/OS batch jobs are typically optimized to run as quickly as possible. When a batch job is moved into CICS, it may have to share resources with online applications. These resources can be CPU, files or databases. It is therefore possible that the batch job may take longer to complete. At the same time, with the removal of the batch window, it may be that the batch job can start earlier or complete later. For batch jobs with hard time deadlines, investigation would be required to understand if the job can complete in the time required.

2.4.2 Implications on online application performance need to be factored

It is important that the batch processing does not negatively impact the performance of online applications. A user invoking a CICS transaction might expect a response time of tenths of a second. If the batch application locks too many resources at a time, it could impact the performance of the online applications. The checkpoint interval of the batch job can be adjusted to change the number of updates made within a checkpoint. If the batch job consumes too much CPU, this can also affect the online applications performance.

2.4.3 Data being updated by batch can be changed by the online applications

Batch applications that rely on a set of records to remain consistent may not work when run in parallel with online applications, because the online applications can update records that the batch job has read, or is about to read. This would need to be considered on a per application basis to determine if it is likely to be an issue.

2.4.4 Traditional batch jobs need refactoring

If you aim to move an existing batch job into CICS, the job needs refactoring to fit into the batch programming model. An overview of the batch programming model is provided in 3.1, “Batch job step” on page 10.

3

Key components of a Batch Environment application

In this chapter, we introduce the key components of a Batch Environment application that are relevant to CICS.

3.1 Batch job step

A batch job is made up of one or more steps. A batch job step is effectively a Java program. The Java program must implement an interface provided by the Batch Environment called the *BatchJobStepInterface*. The methods on the interface are called by the Batch Container to pass control to the batch job step.

The method in which business logic is written is *processJobStep()*. When the method is called, the Java program is expected to perform one logical unit of processing; for example, read one input record and update the master files based on that input record. The Batch Container will call the *processJobStep()* method iteratively until all of the processing is completed.

The reason for returning control to the Batch Container after each unit of processing is so that the Batch Container can take care of management issues such as checkpointing the state of the job at regular (user defined) intervals.

3.2 Batch Data Streams

Batch Data Streams provide a level of abstraction between a resource such as a file or a database, and the Batch Job Step that needs to use those resources. Here are some reasons for this level of abstraction:

- Access to any type of resource follows the same well defined interface, so the Batch Job Step could easily be changed to use a different resource type.
- An implementation of a Batch Data Stream is responsible for ensuring that the resource can be recovered to the last checkpoint position after a failure.

  As an example, imagine that the Batch Data Stream has read up to the record with key 10 in a Key Sequenced Data Set. The Batch Data Stream is asked by the Batch Container to provide recovery information for a checkpoint and it tells the Batch Container that the last record it read had a key of 10.

  Further processing occurs and the Batch Data Stream has read records 11, 12 and 13, when an error occurs. The Batch Container recovers to the last checkpoint. It tells the Batch Data Stream that it must recover to key 10. The Batch Data Stream takes care of that recovery, so the next time the Batch Job Step asks for a record, it is returned record 11. The Batch Job Step remains unaware of the checkpointing and recovery as it is abstracted into the Batch Data Streams.

3.3 XML Job Control Language (xJCL)

xJCL is similar in purpose to the z/OS Job Control Language (JCL). xJCL describes everything about the job, including a sequence of job steps, the batch data streams used by each job step, and the resources associated with each batch data stream.

4

Application design considerations

In this chapter, we highlight some of the key application design considerations when building a batch application to run in CICS.

4.1 When to use Batch Data Streams to read input to a batch job

The use of a Batch Data Stream makes sense in the following circumstances:

- The batch job is processing lots of records and needs to exploit checkpoint/restart.
- You want to maintain a logical physical indirection to an input/output source.
- You allow the application to open something, for example, for input/output based on a logical name, and allow the job definition (xJCL) to govern which implementation to use.
- You want to reuse existing Batch Data Stream implementations.
- You want your batch job step's I/O dependencies to consistently be self-documenting through the Batch Data Stream construct in xJCL.

For input data, it is often the case that a large number of records need to be processed. Therefore, the use of Batch Data Streams for input, to enable checkpoint/restart processing, is generally the best option.

4.2 When to use Batch Data Streams for output from a batch job

The documented reasons for why a Batch Data Stream makes sense for input often also apply for output. However, there may be cases where it is more appropriate not to use Batch Data Streams.

For example, if you want to make a call to an existing CICS program that updates a transactional resource and you want to use that CICS program without change, it may be possible to do so. Bear in mind that when you do not use Batch Data Streams, no checkpoint data is stored to recover to a particular position in the resource. Whether that is an issue will depend on the resource type.

For example, updates to records in a VSAM Key Sequenced Data Set (KSDS) where the records are updated by primary key, may not need checkpoint information stored. If the batch program fails, then as long as at restart, the position in the input data is recovered, then updates can be made again to the records in the KSDS.

4.3 Transactional versus non-transactional output

Batch Data Streams in CICS can be classified into one of three categories:

- Some batch data streams map to CICS managed transactional resources.

  These types of streams access the CICS resources through CICS APIs. If updates or deletes of records are made through one of these Batch Data Streams then those updates will be transactional and will be committed when the Batch Container checkpoints the application.

- Some batch data streams provided with the Batch Environment product access resources directly without passing through CICS.

  These include partitioned data set members and files on z/FS. While these batch data streams can be used within a CICS batch application, however the updates and deletes of records within the resources are not transactional. Therefore, if the batch application encounters an error and is backed out to the last checkpoint, updates to the non-transactional resources are not backed out and the data is left in an inconsistent state. If the batch application is restarted from its last successful checkpoint, then the Batch Data Stream may be able to correct the state of the data. For example, when appending new records to the end of a zFS file, the Batch Data Stream could truncate the file to the offset of the last successful checkpoint and then continue with further processing.

- The third type of batch data stream is user written. If the existing Batch Data Stream implementations do not meet your needs, you can write your own implementations.

4.3.1 Syncpoints in existing code

If your existing CICS programs issue explicit SYNCPOINT commands, these programs will need to be modified before being called from a batch application running under the control of CICS. If the program issues a SYNCPOINT, it will break the checkpointing of the batch application and if the application fails and needs to be restarted, this will result in inconsistent results.

Note: The job step and CICS programs called by the job step must not issue the CICS commands SYNCPOINT or SYNCPOINT ROLLBACK. The Batch Container will commit or roll-back the transaction as required, typically when a checkpoint is required.

4.3.2 High availability

The high availability feature of the Batch Container in CICS is that more than one Batch Container can inform the scheduler that it is capable of running a particular batch job. The job scheduler can then dispatch the application to any one of those Batch Containers. From a CICS perspective the Batch Container can be installed into more than one CICS region. If those CICS regions are to be able to run the same job, then they all need access to the input resources and the output resources for the batch application. It may make sense to run CICS regions across multiple LPARs in a parallel sysplex to maintain high availability in the event of a CICS or LPAR failure or an IPL.

4.4 Moving a job step from traditional batch into the CICS batch container

In this section, we introduce some of the design considerations that should be taken into account if you choose to move a component of a traditional batch job into CICS.

In our mock scenario, we have an existing traditional batch program. One of the steps that the batch program runs is taking several hours. While it is running, online transaction processing in CICS is disabled. There is pressure to reduce the batch window and the program has been identified as one which needs to be addressed. The program is written in COBOL and has exclusive access to a VSAM master file to which it applies updates. The program returns a return code that is used in the JCL to determine if the job can continue. The JCL is submitted from IBM Tivoli® Workload Scheduler.

To determine if the business logic performed by the program can run in a CICS Batch Container, we first need to understand the requirements on the batch job:

- Does the job really need exclusive access to the data?

  If the job needs exclusive access, running it inside CICS is not an option.
- Does the job or chain of dependent jobs have a defined start time?

  We need to understand if the job step can be run at a different time if it was not for the batch window, or does the job step rely on other batch jobs that have to start at a specific time?
- Does the job or chain of dependent jobs have a defined stop time?

  We need to understand if the job can run longer if there is no batch window, or if there are specific external time deadlines that must be met.
- Does the job have any prerequisite steps that need exclusive access to the data?

  This will help us understand if the VSAM master file still needs to be taken offline.
- Is the job step to be followed by any other jobs that need exclusive access to the data?

  This will help us to understand if the VSAM master file would need to be taken offline again after running the jobstep.
- What are the impacts to the data if the business logic performed in the job is done in parallel with online transaction processing?

  We need to understand if the job step can function properly given that the online applications could change records that the job has processed or is about to process. This helps us understand the nature of the job step and whether we could consider running online applications and this batch job step in parallel.
- Can the online applications function properly if they run against master data that has not been fully processed by the batch job step?

  We need to understand whether the online applications expect certain tasks to have been performed by batch jobs against the data on which they are operating.

For the purposes of this example, let us assume that the job step is identified as one that can run in parallel with online applications and is a candidate to run in CICS. The job is the last one to apply changes to the VSAM master file before online application processing resumes. Therefore, if it were to be run in CICS, online application processing could resume earlier and the batch window could be shortened.

The next step is to identify a design:

- Is there COBOL code in the batch job that we want to re-use in CICS?

  If yes, then that code will need some modifications to enable it to be called via the CICS API.

- Is there code in CICS (COBOL, PL/I, C, C++, Java) that could be used in the batch job?

  Perhaps you already have some duplication of business logic between batch and CICS that can be eliminated by re-using the CICS code for the batch job step. For example, updating a record in the VSAM master file.

- Do we need Batch Data Streams to represent the input and output data?

  If you have thousands of records, using Batch Data Streams for the input data in combination with checkpointing ensures if the job fails it can be restarted at the last checkpoint. Whether Batch Data Streams are required for output data will depend on the application.

- How can we take the existing business logic of the batch job and put it into a form that meets the batch programming model?

  The easiest approach could be to separate the I/O of the batch job from the business logic and to implement the business logic as a CICS COBOL program, so that the code can be re-used.

- How can we call the new batch job step from JCL?

  See 6.1, “Invoking the BatchStatementSample application from JCL” on page 48.

In our mock scenario, described in Chapter 5., “End-to-end development scenario” on page 17, we determine we can modify the JCL step to invoke the batch job step in CICS. Online transaction processing can be restarted sooner and the batch job step can run in parallel with the online work. The batch window is reduced, data is shared, and some code sharing might be possible. The new job step returns a return code which is used in the JCL to determine if the job can continue. The JCL can still be submitted from Tivoli Workload Scheduler.

5

End-to-end development scenario

In this chapter, we use a simple scenario to review the development steps to create, deploy, and test a batch application running in CICS.

5.1 Description of scenario

In our scenario, the bank account details are held in a CICS managed VSAM non-RLS file. A batch job needs to be run to create a PDF document for each customer, which contains the customer's current balance.

To achieve this, we create a batch program that does the following processing for each account record:

1. Read the account record from the bank account master file through a Batch Data Stream.
2. Call an existing COBOL program that is responsible for ensuring that the account balance is up to date; for example, applying interest accrued on the account.
3. On return to the Java Job Step, generate a PDF document, containing the balance of the account.
4. Store the PDF document in a file on z/FS.

We make the choice to implement this scenario in a CICS batch application because the generation needs to be done while the online CICS application programs can still access the master customer bank account details file. Existing COBOL logic is called to demonstrate the re-use of existing business logic and how to connect from Java to COBOL using CICS APIs. A PDF document is generated in Java to demonstrate a type of function that is easy to do in Java using a readily available package, but could prove more challenging to implement in a language like COBOL.

5.2 Software used in the scenario

We used the following software levels for the scenario:

- WebSphere Application Server for z/OS V8.5
- DB2® for z/OS Version 9.1
- CICS Transaction Server for z/OS V5.1
- CICS TS Feature Pack for Modern Batch v1.0
- IBM Rational® Application Developer V8.5

Note: CICS TS V5.1 provides the IBM CICS Explorer® SDK (Software Development Kit) which can be used to develop CICS Java applications with the Enterprise Developer Tools - WebSphere batch component and the Java EE Connector - J2C components installed.

In this instance, we chose not to use the CICS Explorer SDK. Instead, we used the modern batch tooling in Rational Application Developer because this tooling is designed specifically for developing Batch Environment applications.

5.3 Setting up the prerequisites for the sample application

The sample batch application developed in this chapter uses a VSAM file containing mock customer bank account details. The download for the sample includes a JCL job called CRTVSAM which we ran to create this file. The JCL job CSDDEF is also provided to define the VSAM file to CICS. The CSDDEF job may give return code 4 the first time it is run, as it includes a command to delete a CSD group called BTCHSAMP and this group may not exist. The job then creates definitions within the BTCHSAMP group. Having run the job, you will need to install the BTCHSAMP group on your CICS region before running the sample.

See “Using the Web material” on page 54 for the source files and instructions for this sample.

5.3.1 Creating the sample VSAM file

To create the input file used by the Redbook sample application, update and submit the `<INSTALL_HLQ>.BTCHSAMP.JCL(CRTVSAM)` JCL job:

1. Change all instances of `<BTCHSAMP_INST>` to the IBM MVS™ high-level qualifier for the batch sample.
2. Review the JCL job card and update as appropriate.
3. Save your changes and submit the job. The job should complete with a return code of 0.

The following data sets are created:

- `<BTCHSAMP_INST>.BTCHSAMP.VSAMMSTR` – Sample VSAM input file
- `<BTCHSAMP_INST>.BTCHSAMP.VSAMMSTR.DATA`
- `<BTCHSAMP_INST>.BTCHSAMP.VSAMMSTR.INDEX`

5.3.2 Adding the sample resources to the CICS CSD

To run the batch sample, you must have resources installed in the CICS region. These resources are supplied in a group called `BTCHSAMP`. You can change the name of the group if it is inappropriate for your environment.

To create group `BTCHBSAMP`, update and submit the JCL job `<INSTALL_HLQ>.BTCHSAMP.JCL(CSDDEF`:

1. Change all instances of `<BTCHSAMP_INST>` to your install high level qualifier.
2. Change `<YOUR_CSD>` to the data set name of your CICS CSD.
3. Change `<CICS_HLQ>` to the high level qualifier of your CICS install data sets.
4. Review the JCL job card and change it as appropriate.
5. Save your changes and submit the job.

5.4 Developing the batch application

This section of the chapter shows how we used Rational Application Developer (RAD) to create a batch application for CICS:

1. Start RAD and, from the main menu, select **File** → **New** → **Batch Project**, as shown in Figure 5-1.

> **Note:** If the Batch Project is not shown on the **File** → **New** list of options, choose **File** → **New** → **Other**, then from the dialog box shown, select **Modern Batch** → **Batch Project**. Then the Batch Project will be in the list.

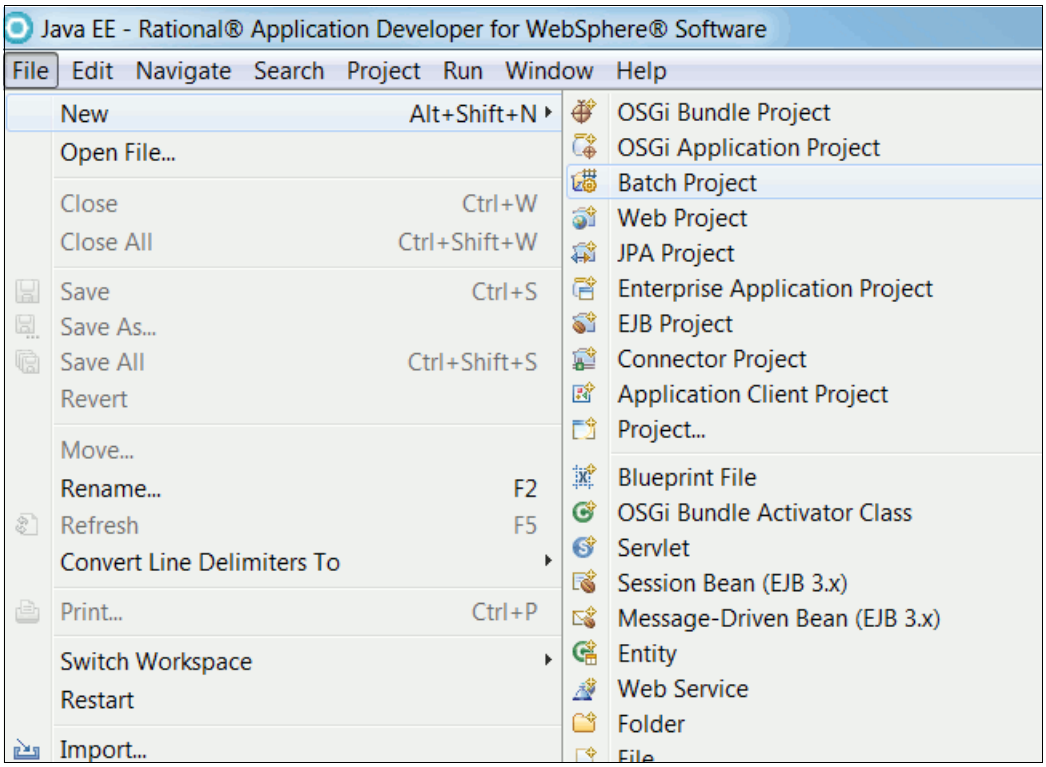

Figure 5-1 Creating a new Modern Batch Project using RAD

2. In the displayed Batch Project dialog box, enter the project name. We called our project CreateStatements. We chose a Target runtime of WebSphere Application Server v8.5. Select **Finish** to create the project. See Figure 5-2.

> **Note:** RAD does not contain a target runtime for CICS. By choosing the WebSphere Application Server v8.5 target runtime, we ensured that the WebSphere Batch Environment jar files needed to build a batch application were automatically added to the classpath for the project. If we had selected **No Target Runtime**, we would have had to manually add the jars to get our project to compile.

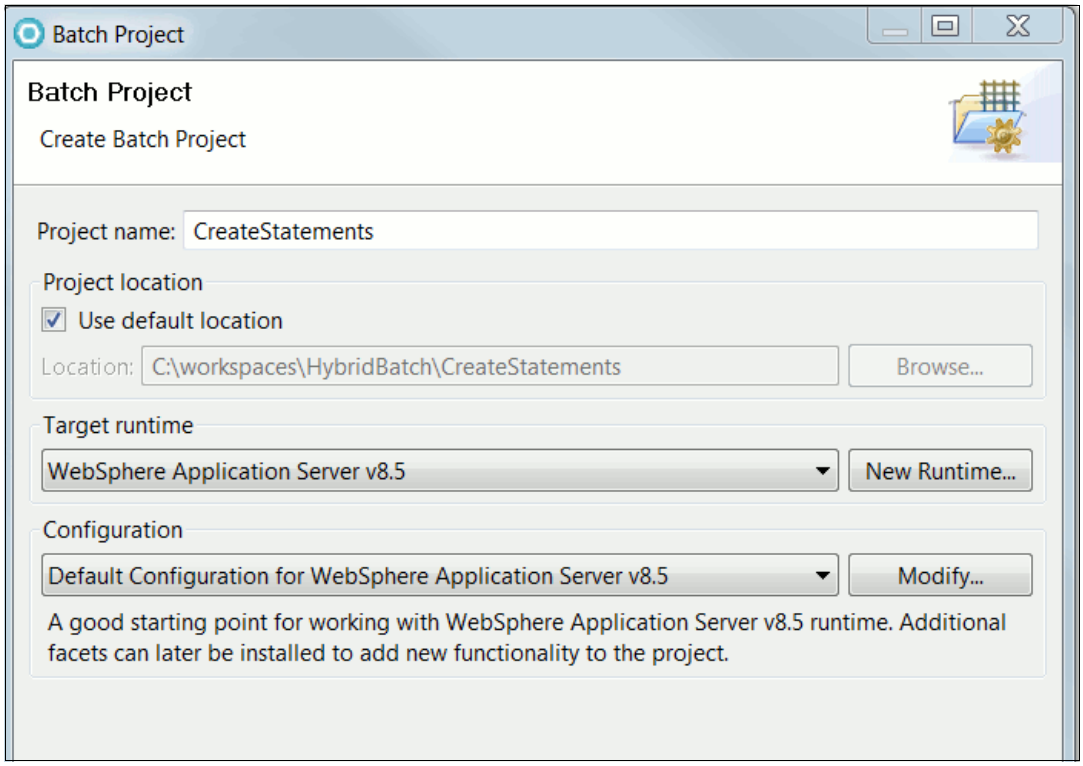

Figure 5-2 Batch Project creation dialog box.

3. RAD creates three projects: an EAR, an EJB, and a standard Java project. For CICS, the only project used will be the standard Java project called CreateStatements. The CreateStatements project contains a directory for Java source and a directory for xJCL files. See Figure 5-3.

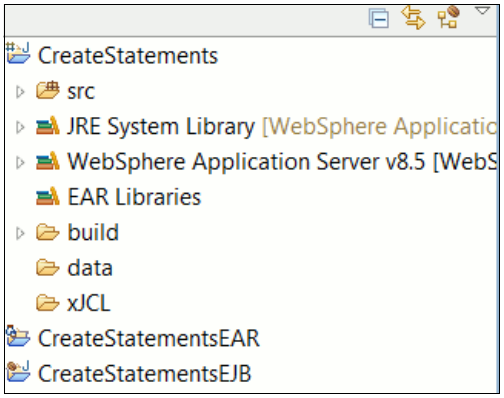

Figure 5-3 Modern batch application projects created by RAD

4. Before proceeding to generate any batch artifacts, we configured the build path for the CreateStatements project. Right-click **CreateStatements** and select **Build Path** → **Configure Build Path**, as shown in Figure 5-4.

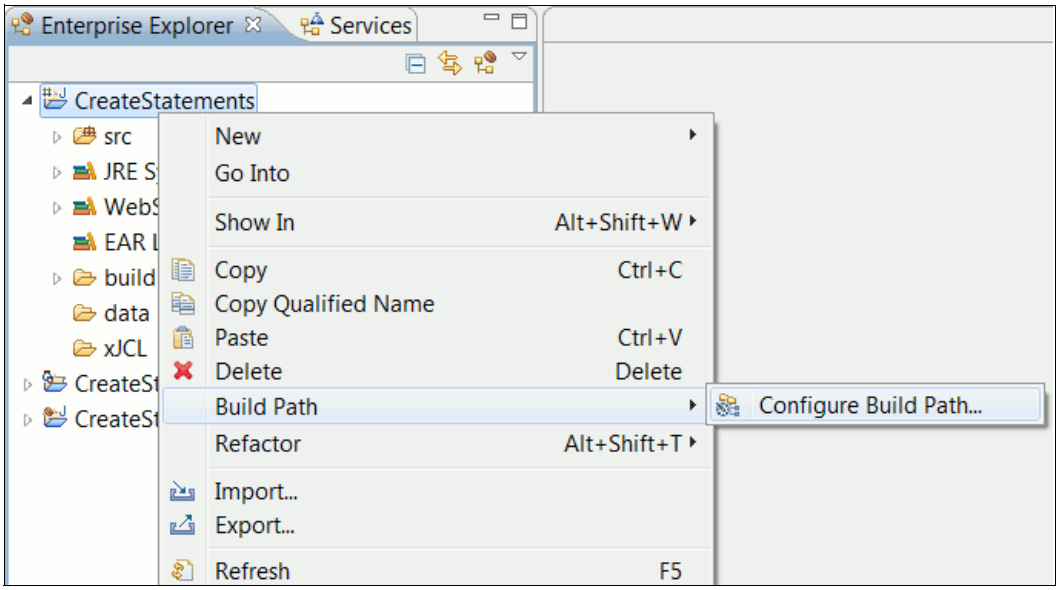

Figure 5-4 Configuring the build path for the CreateStatements project

5. Our example project requires the following additional Jars:
   - JCICS jars for accessing CICS resources
   - CICS TS Feature Pack for Modern Batch jars
   - Jars for creating PDF files

   Select **Add External Jars** and add the jars shown in Figure 5-5. Click **Ok** to complete the modifications.

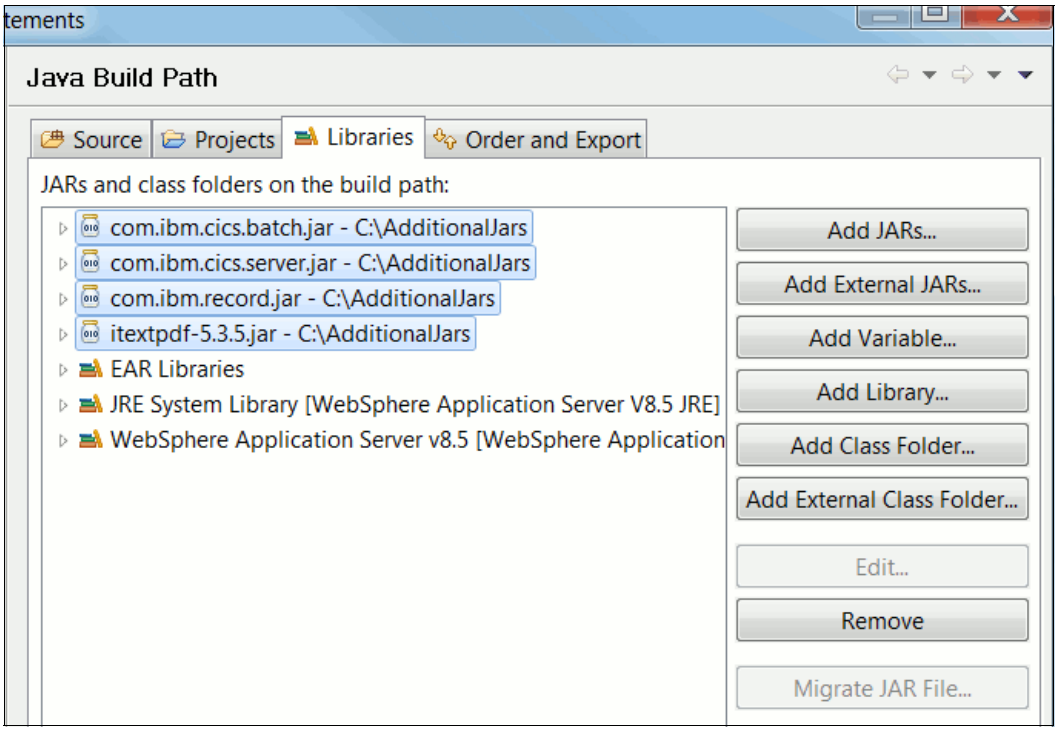

Figure 5-5 Additional Jars required for the example batch job

6. We used a RAD wizard to create a template for our batch job step and to create the xJCL for the batch job. Right-click the **CreateStatements** project and select **New** → **Batch Job** as shown in Figure 5-6.

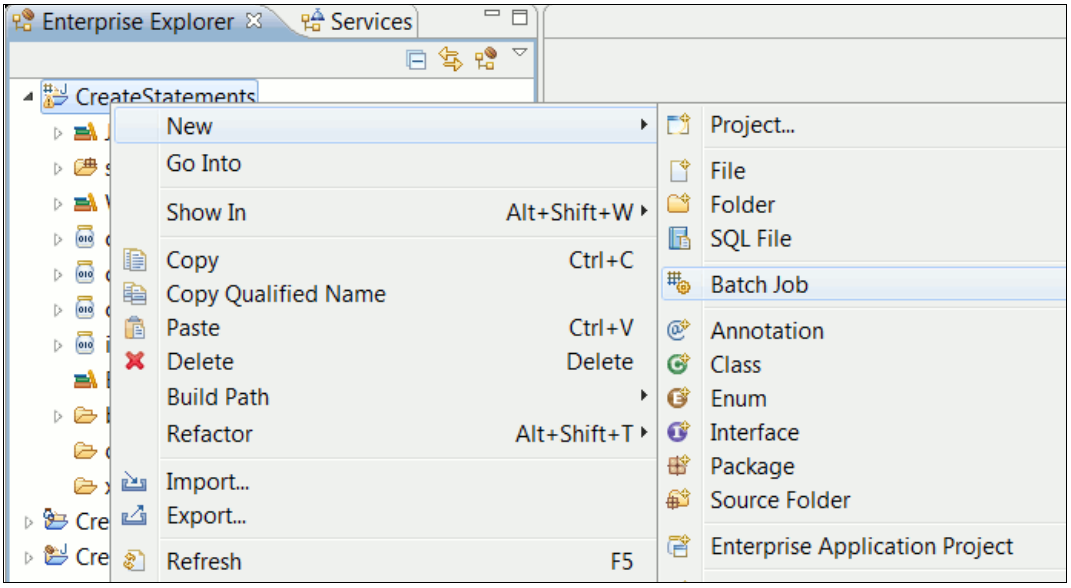

Figure 5-6 Using the RAD wizard to create a new Batch Job

7. The Batch Job Creation wizard starts. Enter the following values:
   - JobType = `Batch`
   - Job Name = `BatchStatementSample`

   Leave the rest of the fields as default values and click **Finish**. See Figure 5-7.

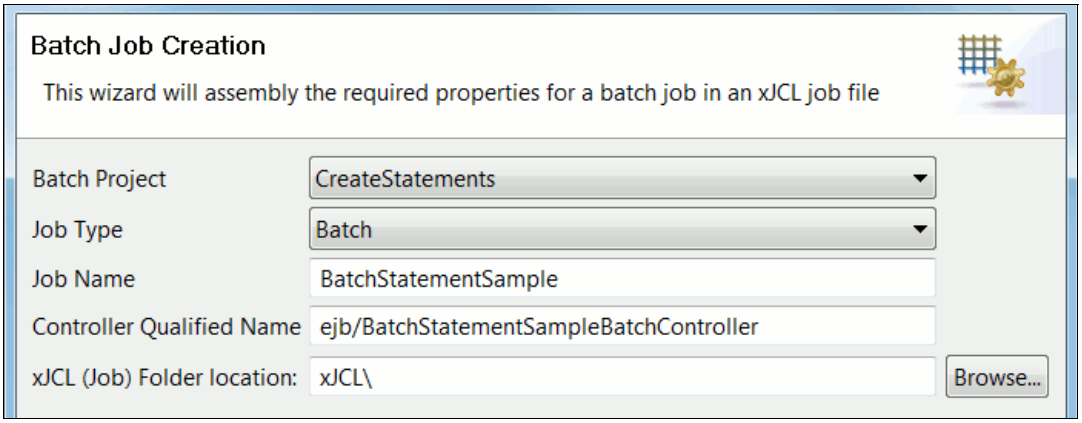

Figure 5-7 Extract of the Batch Job Creation dialog box

8. The next dialog box is for Batch Step Creation. Our example job has one step, we define it here. Enter the following values:
   - Name = `AccountBalancePDFGenerator`
   - Select Pattern = `Custom`

> **Note:** Job steps can be created from patterns. The intent of a pattern is to minimize the code that needs to be written for job steps that follow that pattern. In our example, we chose not to use a pattern, but to create a job step that directly implements the BatchJobStepInterface.

The dialog box indicates an error at this point, highlighting that an implementation class for the batch job step must be specified as shown in Figure 5-8. Click **Create Class** to create the class for the batch job step.

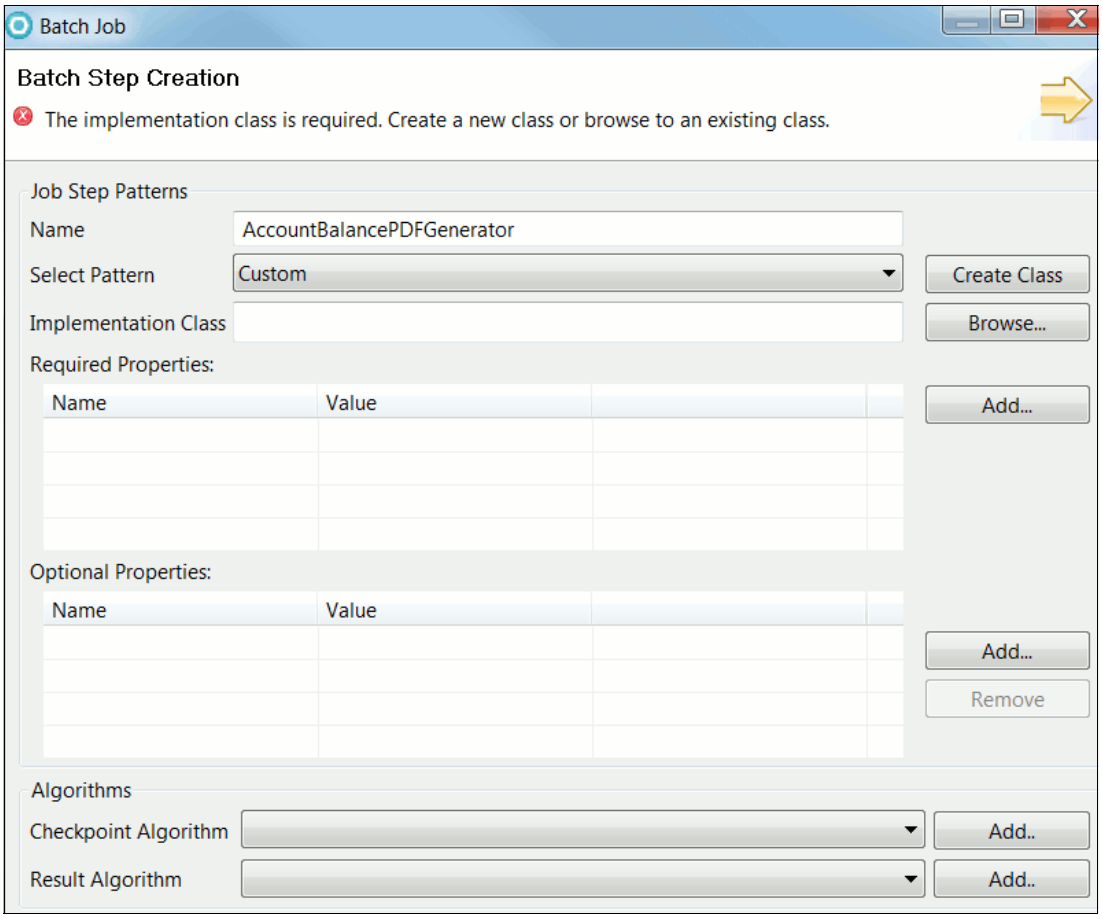

Figure 5-8 The Batch Step Creation dialog box

9. The Create Class dialog box is shown as in Figure 5-9. Enter the following values:
   - Package = `com.ibm.itso.sample`
   - Name = `AccountBalancePDFGenerator`

   Leave the rest of the values as defaults and click **Finish**. RAD creates a Java class called AccountBalancePDFGenerator that implements the BatchJobStepInterface. Later on, we will fill in some of the methods in the AccountBalancePDFGenerator class.

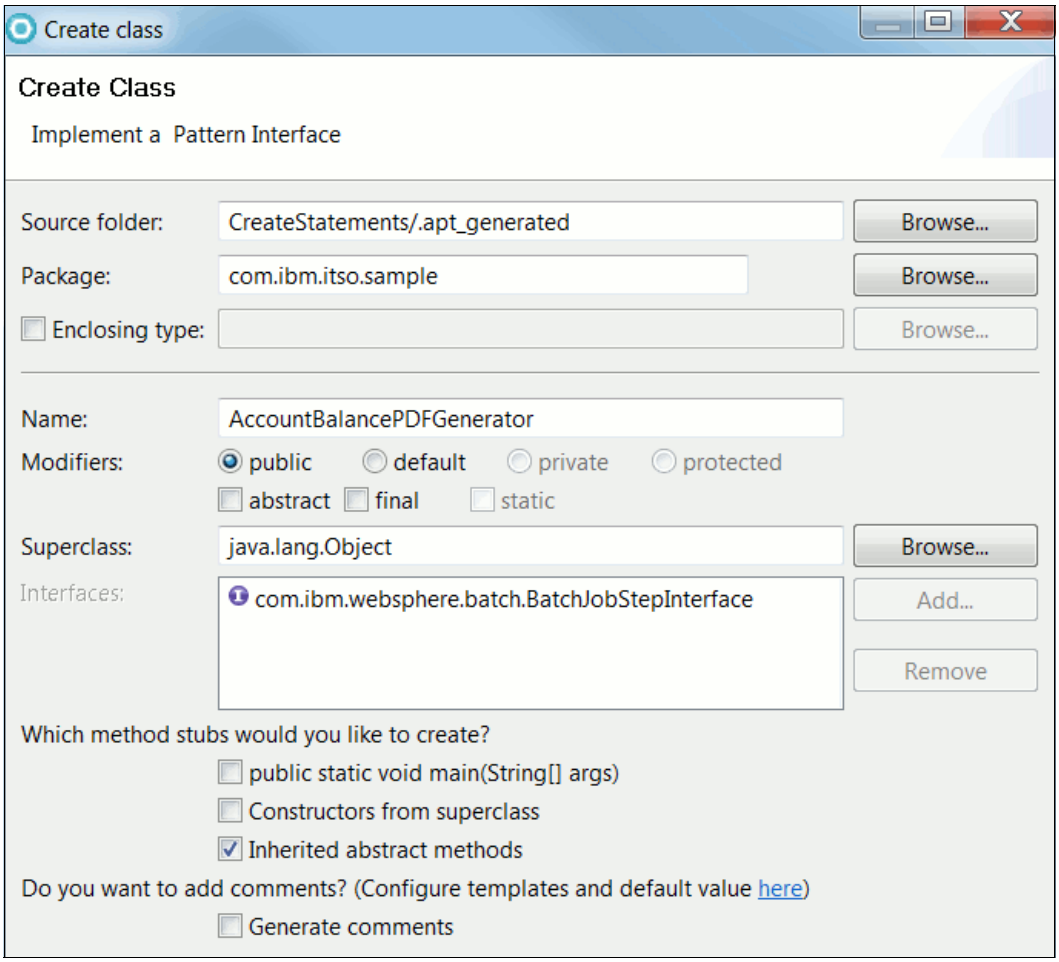

Figure 5-9 Creating a Batch Job Step implementation class

10. Once the implementation class is created, RAD automatically fills in that field in the Batch Step Creation dialog as shown in Figure 5-10. To enable later debug of our batch job step, we specify an optional property on the batch job step. This property will be put in the generated xJCL and can be queried by our batch job step application. Create the optional property debug as follows:
    - debug = `true`

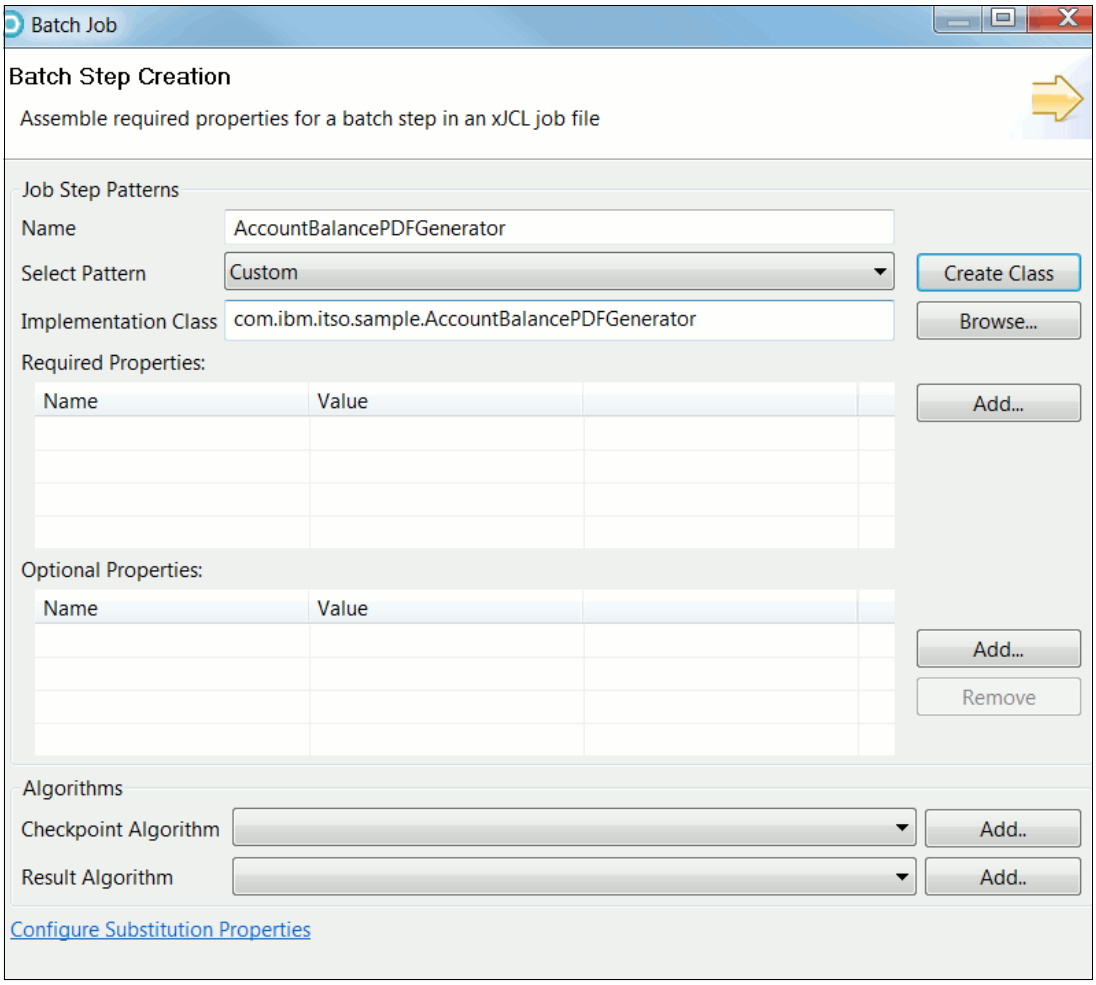

Figure 5-10 Creating an optional property for the Batch Job Step

11. Click the **Add** button to the right of Checkpoint Algorithm to add a checkpoint algorithm to the generated xJCL.

The Checkpoint Algorithm dialog box appears as shown in Figure 5-11. The checkpoint algorithm tells CICS how often to take a checkpoint. We chose to checkpoint based on the number of records that have been processed. As a starting point, we chose to tell CICS to checkpoint every 5 records. This value will be created in the generated xJCL and can be modified before submitting the xJCL to the job scheduler. Enter the following values:

– Name = `chkpt`

– Select Pattern = `Record Based`

Two required properties are displayed in the dialog box for record based checkpointing. Enter the following values:

– TransactionTimeOut = `30`

– recordcount = `5`

The recordcount property is how CICS knows to checkpoint every 5 records. Click **Finish** to create the checkpoint algorithm.

> **Note:** CICS does not support the TransactionTimeOut property and will ignore the value specified; however, a value must be specified or the wizard will not complete.

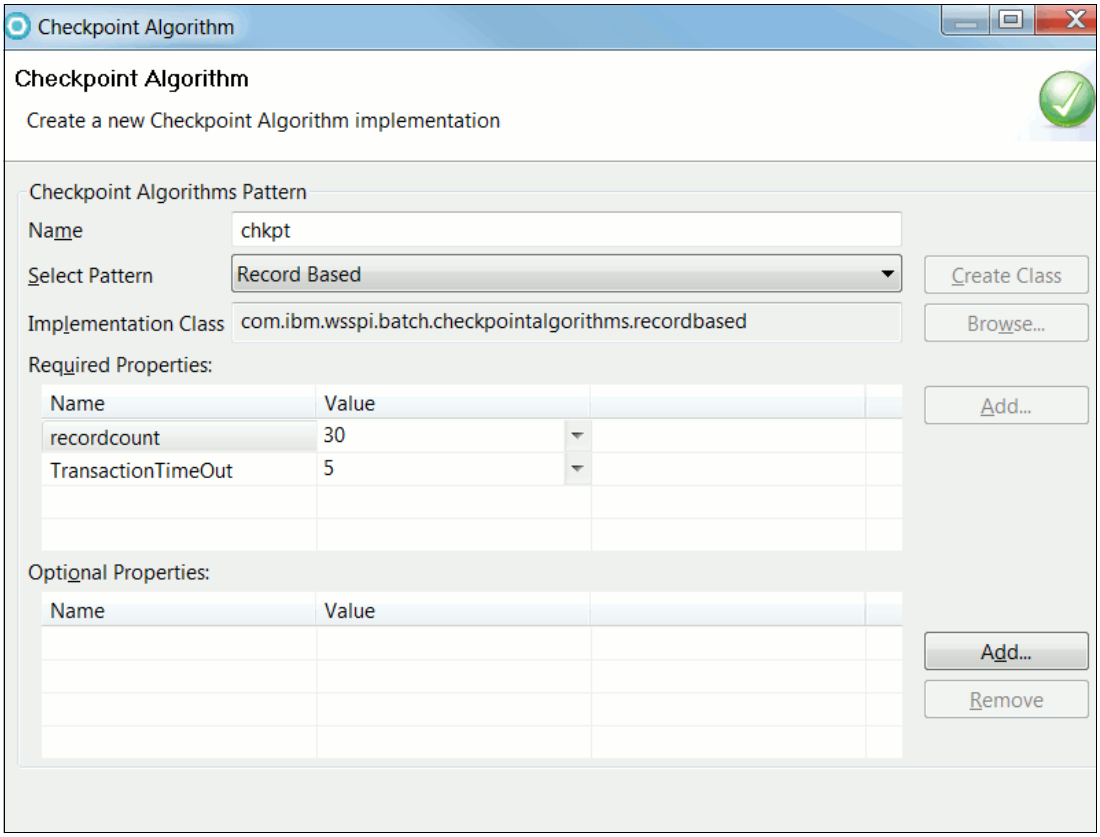

Figure 5-11 Creating a new checkpoint algorithm

12. Click the **Add** button to the right of Result Algorithm to add a result algorithm to the generated xJCL.

The Result Algorithm dialog box appears as shown in Figure 5-12. The result algorithm tells CICS how to calculate the return code for the batch job. We chose to use a provided algorithm called jobsum, which returns the highest return code from any jobstep as the return code for the job. Enter the values:

– Name = `jobsum`

– Select Pattern = `Job Sum`

Click **Finish** to add the result algorithm to the batch job.

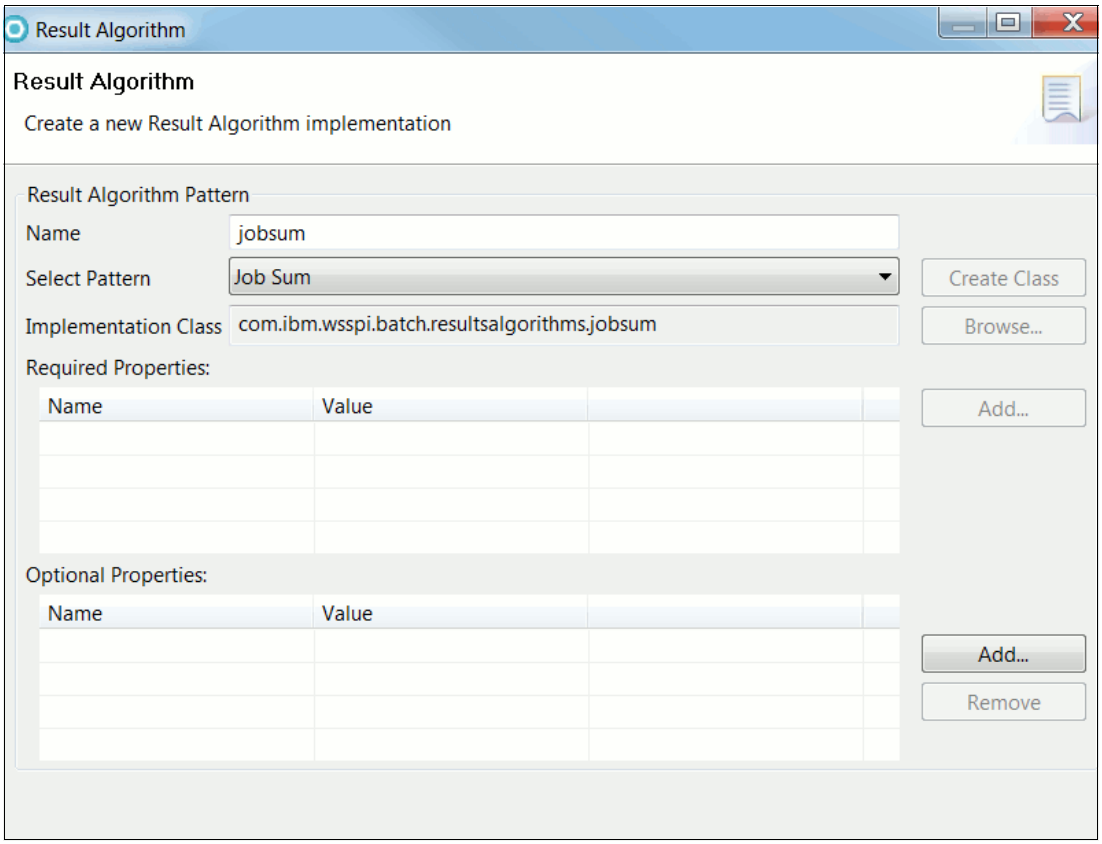

Figure 5-12 Adding a result algorithm to the batch job

13. Figure 5-13 shows the completed Job Step Creation dialog box. Click **Next** to move to the next part of the wizard.

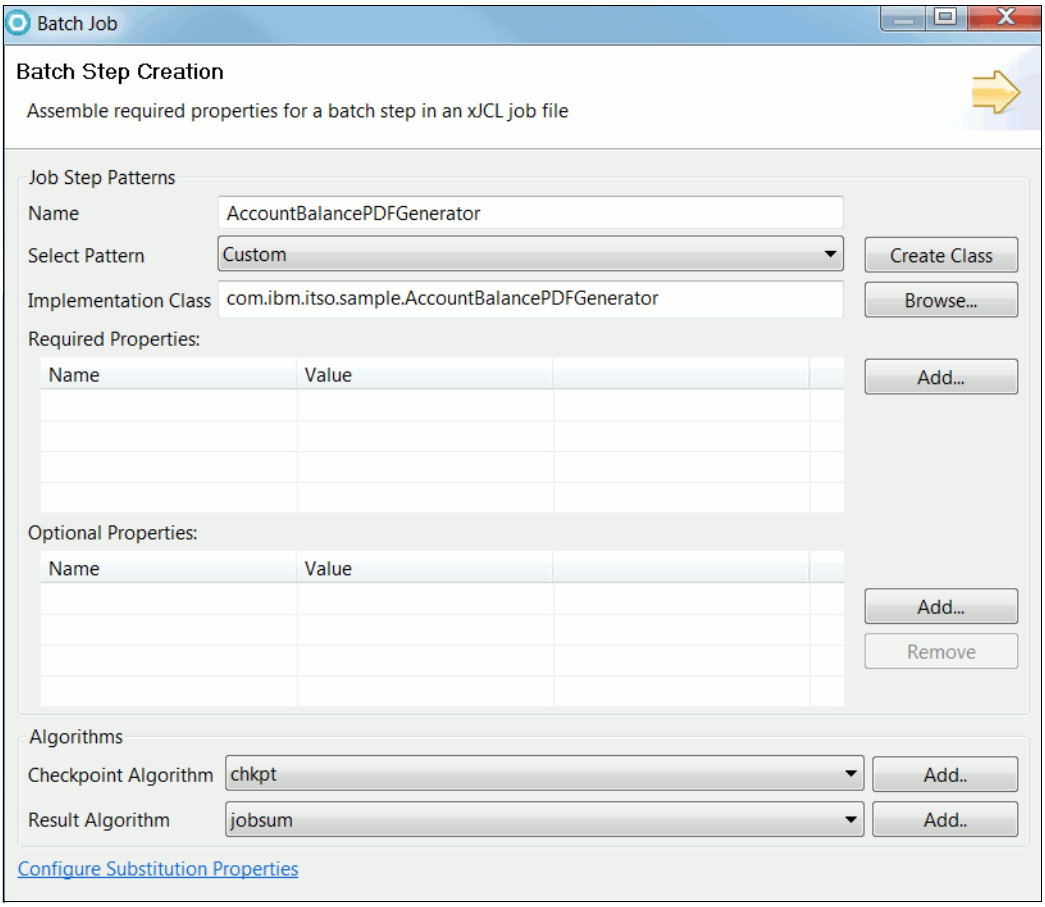

Figure 5-13 The completed Job Step Creation dialog box

14. The next step in the job creation wizard is to define the Batch Data Streams required for the job step. Our example job step uses one Batch Data Stream to read input records from a CICS managed VSAM key sequenced data set. The Batch Data Stream for reading the records is provided as part of the CICS TS Feature Pack for Modern Batch. In the Step Stream dialog box, enter the values:

   – Name = inputStream

   – Select Pattern = Custom Stream

   > **Note:** RAD is unaware of the Batch Data Streams provided with the CICS TS Feature Pack for Modern Batch, so a Custom Stream must be specified in the dialog box.

   – Implementation class = com.ibm.cics.batch.bds.impl.VsamKsdsReaderImpl

> **Important:** The name specified for the batch data stream is a logical name used in the xJCL. In the batch job step implementation class, the logical name is used to obtain a reference to an instance of the batch data stream class.

The CICS provided batch data stream requires three properties so that it knows which CICS file to access, the length of the primary key in that file, and a starting value for the primary key. The starting value for the primary key can be used to specify the point in the file from which to start reading records. In the Required Properties section, enter the values:

- CICSFILE = `SAMPIN`
- KEYLENGTH = `8`
- START = `F0F0F0F0F0F0F0F0`

> **Note:** The START property tells CICS which record to start from. CICS needs a binary key, but xJCL contains only character data, so the key value is encoded in hexadecimal. `F0F0F0F0F0F0F0F0` is a hexidecimal key of numeric zeros, so CICS will start reading records from the beginning of the file.

An example of this dialog box is shown in Figure 5-14. No output Batch Data Streams are required for this example, so click **Finish** to complete the wizard for batch job creation.

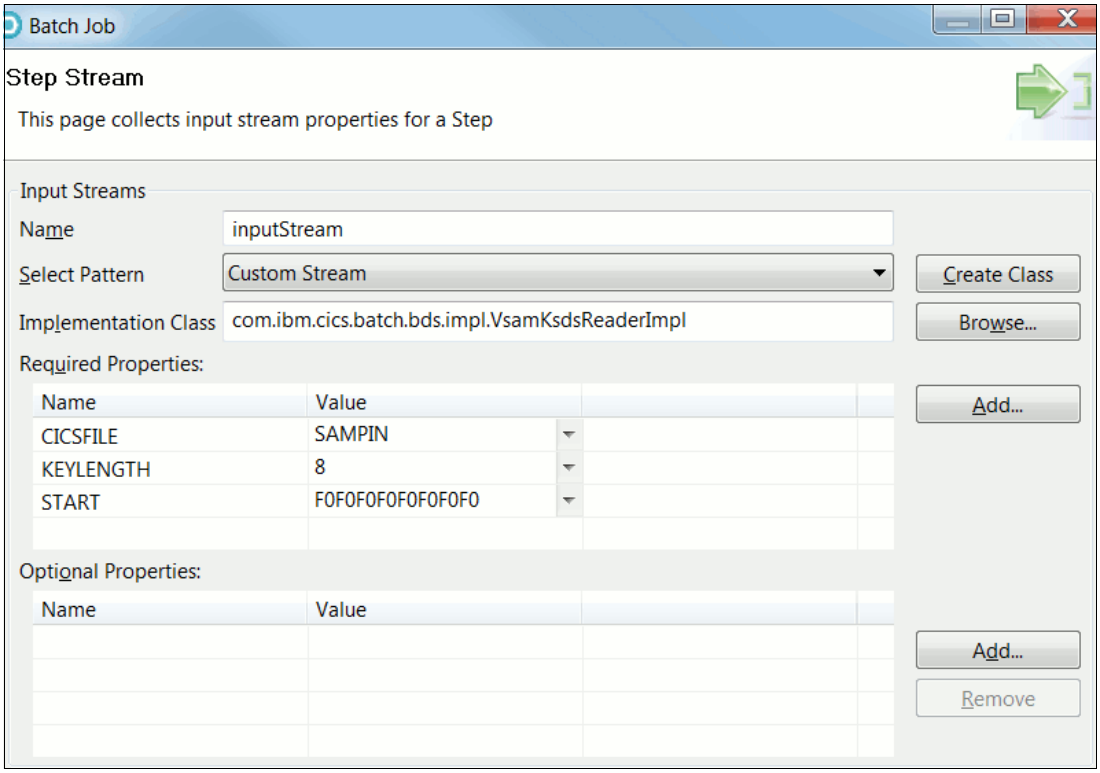

Figure 5-14 Creating a Batch Data Stream definition

Figure 5-15 shows an extract from the xJCL that is created by the wizard. The xJCL can be found in the xJCL directory of the CreateStatements project.

```
AccountBalancePDFGenerator.java    BatchStatementSample.xml
<?xml version="1.0" encoding="ASCII"?>
<job xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespac
  <job-type>Batch</job-type>
  <jndi-name>ejb/BatchStatementSampleBatchController</jndi-name>
  <step-scheduling-criteria>
    <scheduling-mode>sequential</scheduling-mode>
  </step-scheduling-criteria>
  <checkpoint-algorithm name="chkpt">
    <classname>com.ibm.wsspi.batch.checkpointalgorithms.recordbased</clas
    <props>
      <prop name="recordcount" value="30"/>
      <prop name="TransactionTimeOut" value="5"/>
    </props>
  </checkpoint-algorithm>
  <results-algorithms>
    <results-algorithm name="jobsum">
      <classname>com.ibm.wsspi.batch.resultsalgorithms.jobsum</classname>
    </results-algorithm>
  </results-algorithms>
  <job-step name="AccountBalancePDFGenerator">
    <classname>com.ibm.itso.sample.AccountBalancePDFGenerator</classname>
    <checkpoint-algorithm-ref name="chkpt"/>
```

Figure 5-15 Extract of the generated xJCL

Figure 5-16 shows an extract of the generated Java class for the Batch Job Step. This class is where the code for the job step will be added.

```
AccountBalancePDFGenerator.java    BatchStatementSample.xml
package com.ibm.itso.sample;

import java.util.Properties;

public class AccountBalancePDFGenerator implements BatchJobStepInterfac

    @Override
    public void createJobStep() {
        // TODO Auto-generated method stub

    }

    @Override
    public int destroyJobStep() {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public Properties getProperties() {
        // TODO Auto-generated method stub
        return null;
```

Figure 5-16 An extract of the generated Batch Job Step implementation class

15. The next step in creating the batch application is to modify the generated batch job step implementation class and add business logic for the job step. The full source for the job step is provided as a download. Figure 5-17 shows an extract from the processJobStep() method of the AccountBalancePDFGenerator class.

AccountBalancePDFGenerator.java

```
//Use JCICS to call a COBOL program to perform the business logi
Task task = Task.getTask();
Channel channel = task.getCurrentChannel();

//Pass the input record
Container inputContainer = channel.getContainer("INPUT-DATA");
if (inputContainer == null) {
  inputContainer = channel.createContainer("INPUT-DATA");
}
inputContainer.put(inputRecordString);

//Indicate that the COBOL program is about to be called
if (logger.isDebugEnabled()) {
  logger.debug("Calling COBOL business logic");
}

//Link to the COBOL program
Program cicsProgram = new Program();
cicsProgram.setName("BALUPDAT");
cicsProgram.link(channel);

//Read the updated record
Container outputContainer = channel.getContainer("OUTPUT-DATA");

String newCatalogRecordString = "ERROR";
if (outputContainer != null) {
```

Figure 5-17 Extract from the Java code for the Batch Job Step

The code in Figure 5-17 shows a call from Java into an existing CICS COBOL program. The COBOL program expects to receive data in a container on a channel and returns data in another container on the channel. The Java code stores the input data in a container named INPUT-DATA. It issues a link to the CICS program BALUPDAT, then reads the output data from a container named OUTPUT-DATA. The corresponding COBOL program is provided as part of the Redbooks publication examples download.

One of the challenges when passing data between languages such as COBOL and Java, is that a data structure described by a COBOL copybook will typically be passed to the Java program as an array of bytes. There are two technologies supported by CICS TS V5.1 that assist the programmer in accessing the data from a byte array. These technologies are JZOS, which is included as a part of the IBM Java SDK for z/OS, and the J2C capability of RAD. In both cases, a COBOL copybook describing the data to be passed to the Java program is provided to the tooling, and the tooling generates code that makes it easy to access the various fields in the data from Java.

For this example, we chose to use the J2C tooling in RAD. The tooling generated a Java class called CustomerAccount which is provided with the sample code. A step-by-step guide for using both JZOS and the J2C approaches is included in the Redbooks publication, *Java Application Development for CICS,* SG24-5275.

5.5 Testing the example application

While our setup for RAD contains a unit test environment for Batch Environment applications, the fact that the job step we created contains JCICS calls, means that it can only be tested in a CICS batch environment.

Here is an overview of the steps that we follow:

1. Export the application from RAD.
2. Install the application into CICS.
3. Notify the job scheduler that CICS can run the batch job.
4. Submit the xJCL to run the batch job.

5.5.1 Export the application from RAD

We follow these steps:

1. To export the application from RAD, we need to create a JAR file containing the compiled Java classes. From within the CreateStatements project view, expand **src** then right-click **com.ibm.itso.sample** and select **Export** from the context menu, as shown in Figure 5-18.

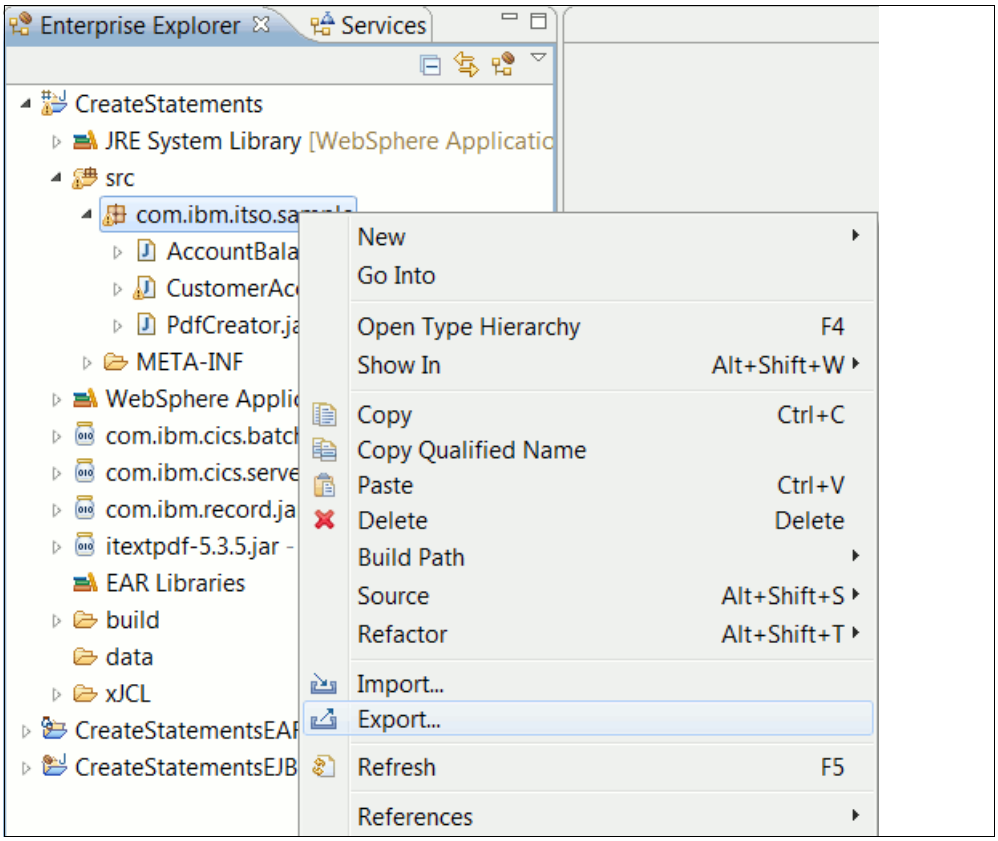

Figure 5-18 Exporting Java code into a Jar file

2. From the Export dialog box, choose **JAR file** from within the Java section, as shown in Figure 5-19, then click **Next**.

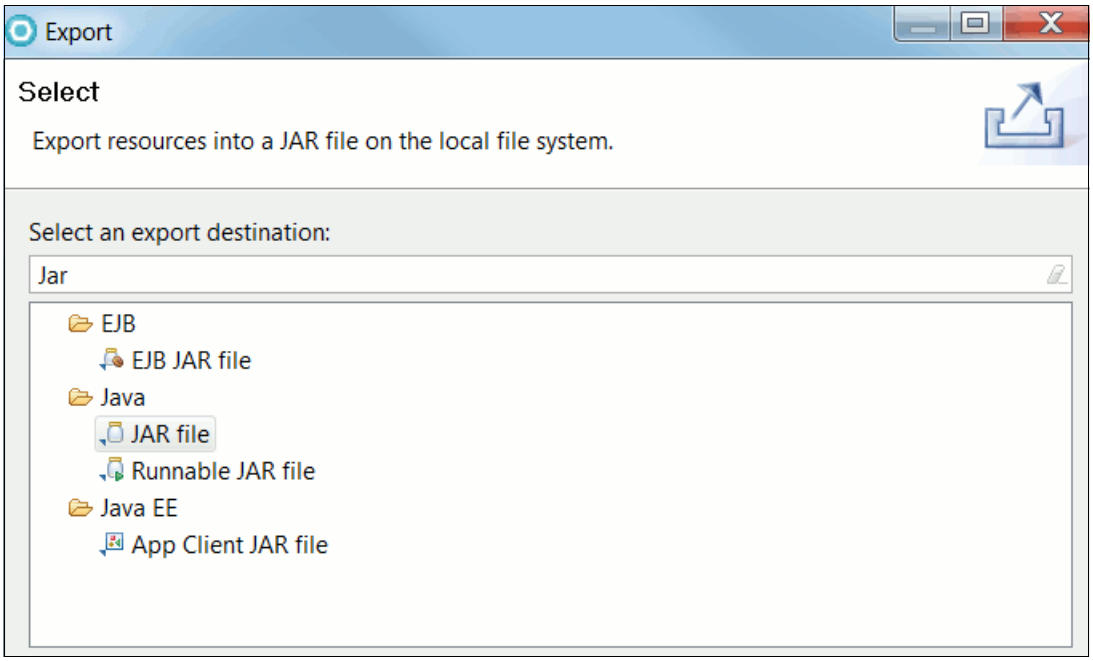

Figure 5-19 Exporting to a Jar file

3. Choose the export destination directory and filename for where you want to create the jar file, as shown in Figure 5-20 and click **Finish**. The Jar file will be created.

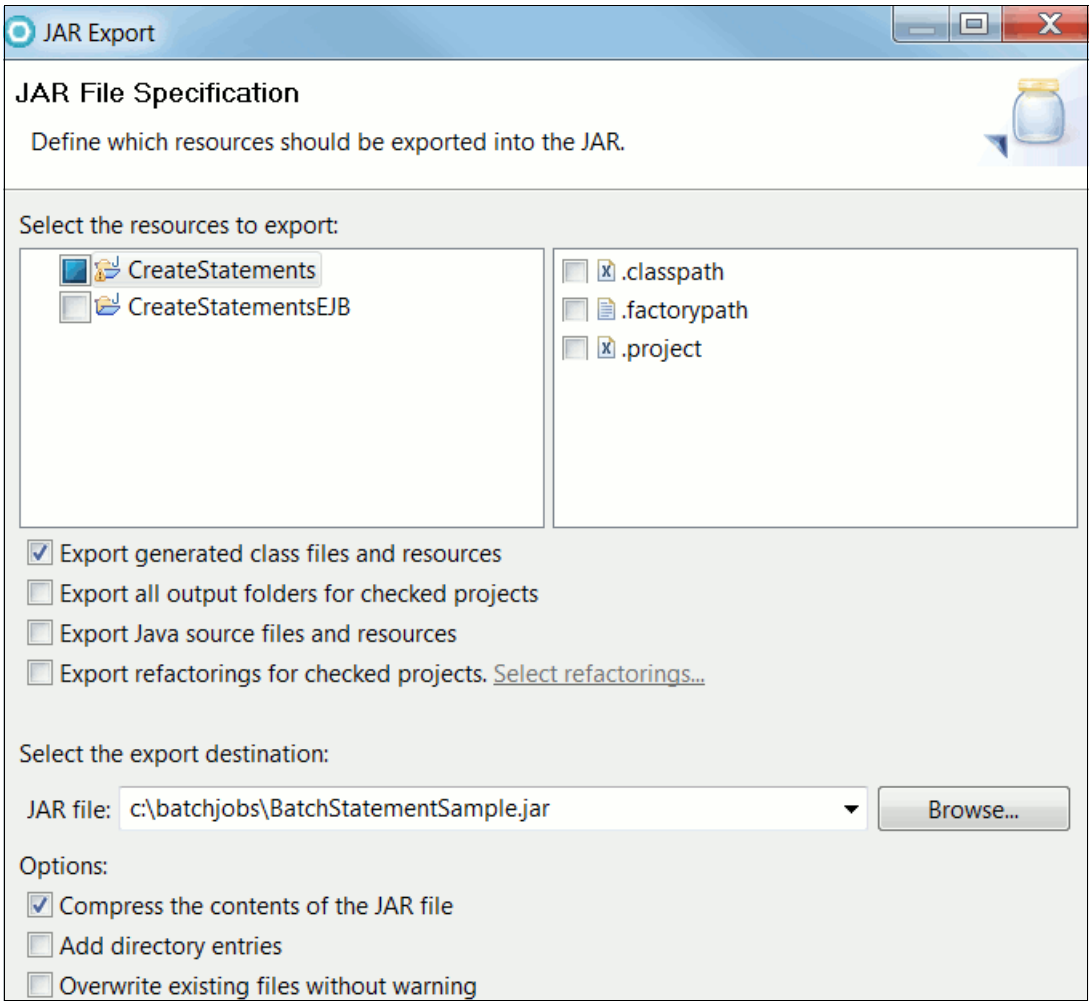

Figure 5-20 Specifying the filename for a Jar file export

5.5.2 Installing the application into CICS

Follow these steps to install the application into CICS:

1. Copy the JAR file from your workstation onto the mainframe. We used ftp to transfer the file in binary mode. Store the file in a directory that CICS has permissions to access. Ensure that CICS has permission to read the JAR file too.
2. Modify the JVM profile for the JVM server that the Batch Container is running in. For our example application, we had to add the following JARs to the CLASSPATH_SUFFIX (Figure 5-20):
   - `BatchStatementSample.jar` - This contains our sample application classes.
   - `marshall.jar` - This is used by the generated J2C class for converting data between COBOL and Java.
   - `j2ee.jar` - This is used by the generated J2C class for converting data between COBOL and Java.
   - `itextpdf-5.1.5.jar` - This contains the open source PDF generation classes.

```
CLASSPATH_SUFFIX=@batchinst@/lib/com.ibm.cics.batch.jar:\
                 @batchinst@/lib/batchframework.jar:\
                 @batchinst@/lib/pgccommon.jar:\
                 @batchinst@/lib/pgcruntime.jar:\
                 @batchinst@/lib/pgcstandalone.jar:\
                 @db2inst@/classes/db2jcc4.jar:\
                 @db2inst@/classes/db2jcc_license_cisuz.jar:\
                 @batchinst@/lib/com.ibm.cics.batch.samples.jar:\
                 @batchinst@/lib/BatchStatementSample.jar:\
                 @batchinst@/lib/itextpdf-5.3.5.jar:\
                 @batchinst@/lib/j2ee.jar:\
                 @batchinst@/lib/marshall.jar:\
```

Figure 5-21 The CLASSPATH_SUFFIX in the JVMProfile of the JVMServer

3. The CICS JVMSERVER resource will need to be disabled and re-enabled to pick up the changes to the classpath.
4. After re-enabling the JVMSERVER, run the CBCH transaction if it is not already running. See “Validation of the CICS connection to Batch Environment” on page 44.

5.5.3 Notifying the scheduler that CICS can run the batch job

The CICS TS Feature Pack for Modern Batch includes a file named `batchcontainer-config.xml`. This file is used to configure the Batch Container and to tell CICS how to contact the job scheduler. One component of the file is a list of logical batch application names that CICS can run. This list of batch application names needs updating to include the sample application.

1. A Sample batchcontainer-config.xml file is located at the `<CICS_INSTALL>/batchfp/samples/config/` directory on z/FS. By default, the batch container will look for this in the `JVMProfiles` directory specified to CICS, otherwise it will use the JVM Profile property **“**`-Dcom.ibm.cics.batch.config`**”** if specified, to look in an alternative location.
2. Edit the `batchcontainer-config.xml`. To add a batch application to the list of supported applications, add the highlighted XML into the `<InstalledApplications>` section of the `batchcontainer-config.xml` file, as shown in Example 5-1.

Example 5-1 Updated section of batchcontainer-config.xml file

```
<InstalledApplications>
     <BatchApplication>
      <name>CICSBatchSample</name>
      <status>enabled</status>
     </BatchApplication>
     <BatchApplication>
      <name>BatchStatementSample</name>
      <status>enabled</status>
     </BatchApplication>
</InstalledApplications>
```

3. Save the changes to the batchcontainer-config.xml file. The CICS Batch Container sends heartbeat information to the job scheduler at regular intervals. This includes information on the applications that the Batch Container can run. At the next heartbeat, the scheduler will be told that CICS can run the batch job called BatchStatementSample.

5.5.4 Submitting the xJCL to run the batch job

In this section, we use the Job Management Console provided by the Batch Environment to submit the xJCL for the application. Later in the chapter, we demonstrate how to submit the same job from traditional JCL:

1. In the previous section, we configured CICS via the batchcontainer-config.xml file, to advertise that it could run a job called BatchStatementSample. Before submitting the xJCL, we modify it so that the default application name it specifies is also named BatchStatementSample. From the Project Explorer in RAD, expand **CreateStatements** → **xJCL** and double-click **BatchStatementSample.xml**, as shown in Figure 5-22. The xJCL is opened in the xJCL editor.

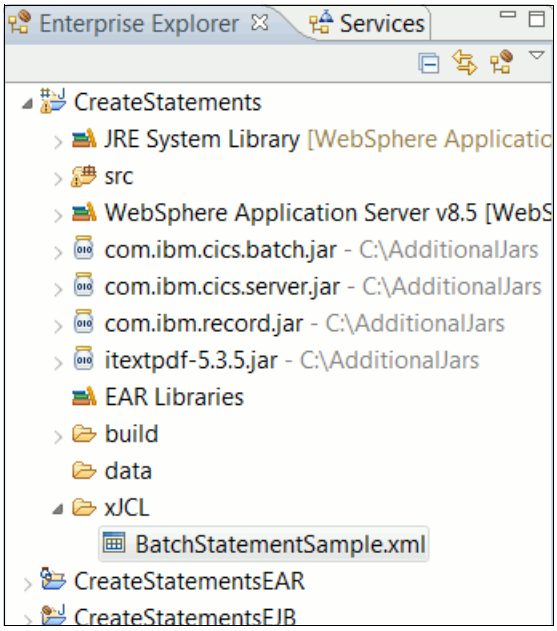

Figure 5-22 The BatchStatementSample.xml file

2. Look at the value of the Default Application Name field in the Details section of the xJCL editor. When the xJCL wizard created the xJCL, it set the name to the name of the Enterprise Archive project CreateStatementsEAR, as shown in Figure 5-23.

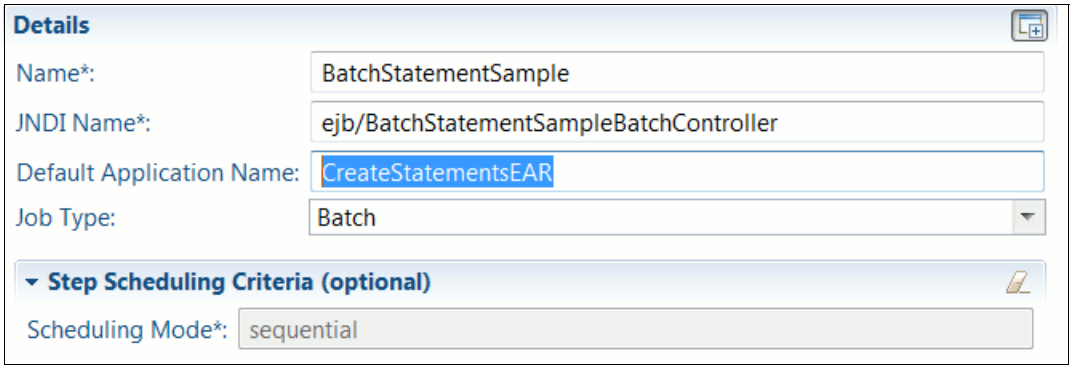

Figure 5-23 Details section from the xJCL editor

3. Modify the Default Application Name to BatchStatementSample, as shown in Figure 5-24, then save the changes. The xJCL is now ready to be submitted.

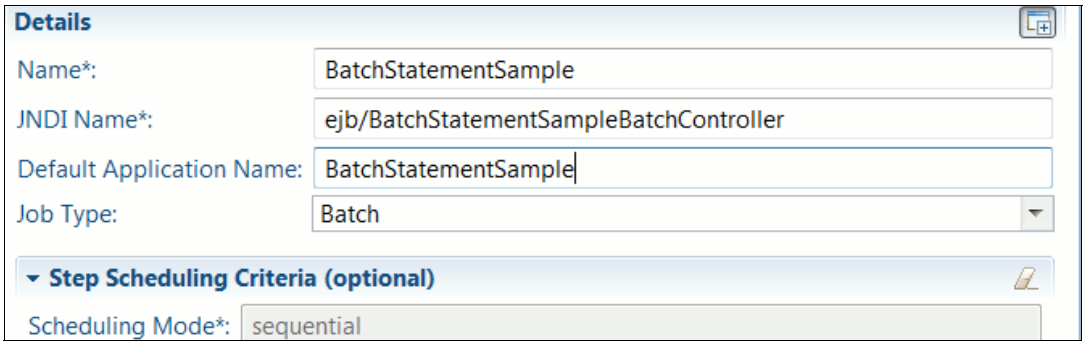

Figure 5-24 Modifying the Default Application Name

4. Make a note of the directory in which the xJCL is stored, or drag and drop it into another directory from where you can easily find the file later.
5. From a web browser, connect to the Job Management Console and logon if necessary. You should be presented with a menu similar to Figure 5-25.

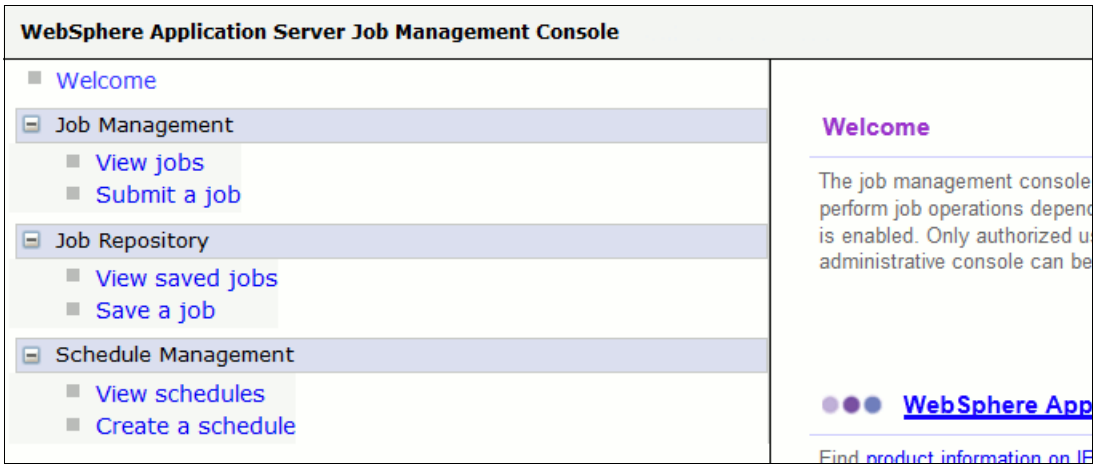

Figure 5-25 Extract from the Job Management Console

6. To submit a job, select **Job Management** → **Submit a job**. Select the **Local file system** and specify the path to the xJCL on your local workstation, as shown in Figure 5-26. Select **Submit** to submit the xJCL to the job scheduler.

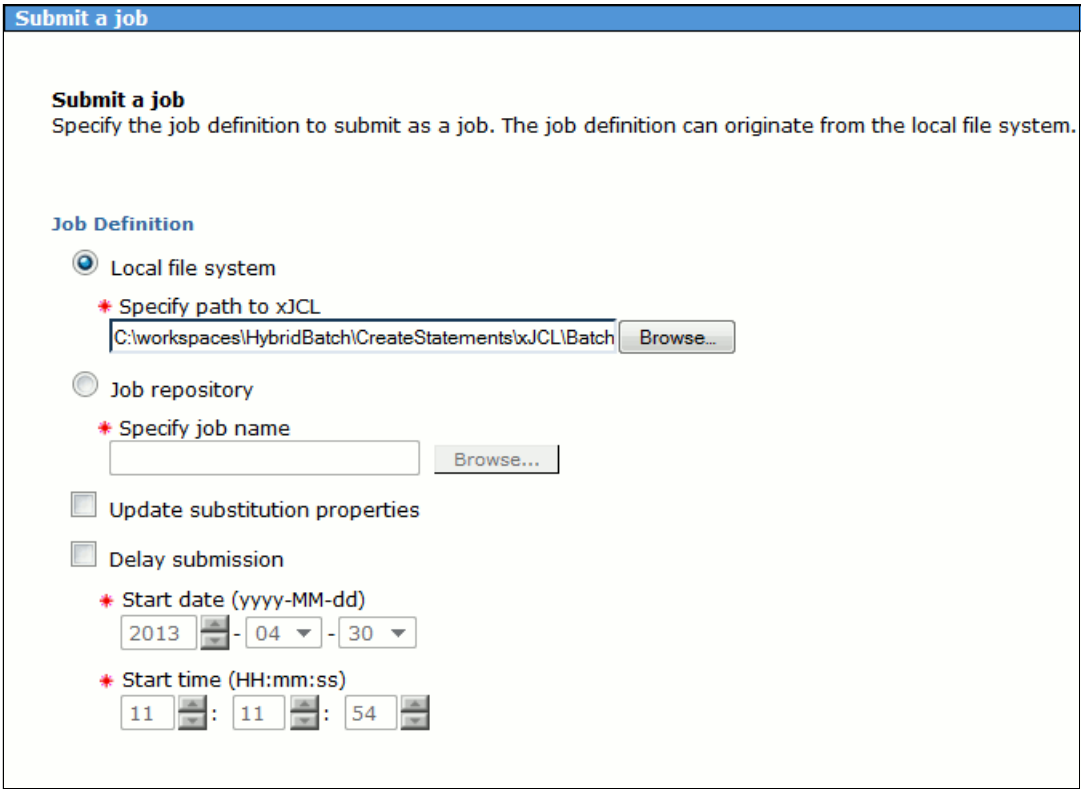

Figure 5-26 Submitting a job from the Job Management Console

7. A message is shown, which confirms that the job has been submitted successfully and provides the job ID, as shown in Figure 5-27. Make a note of the job ID as you will use this to identify the job output.

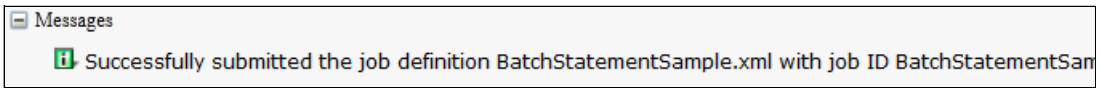

Figure 5-27 Job submitted message showing job ID

Note: If the job is unable to be submitted, see “Debugging a batch job step” on page 43 for suggestions on how to debug problems.

8. When a job is successfully submitted, it means the job scheduler has identified at least one Batch Container which claims to be able to run the job. The scheduler will choose a Batch Container and dispatch the job to it. We can examine the log for the job to see which Batch Container the scheduler has chosen to dispatch the job to. From the Job Management Console, select **Job Management** → **View jobs**. A list of jobs is displayed as shown in Figure 5-28.

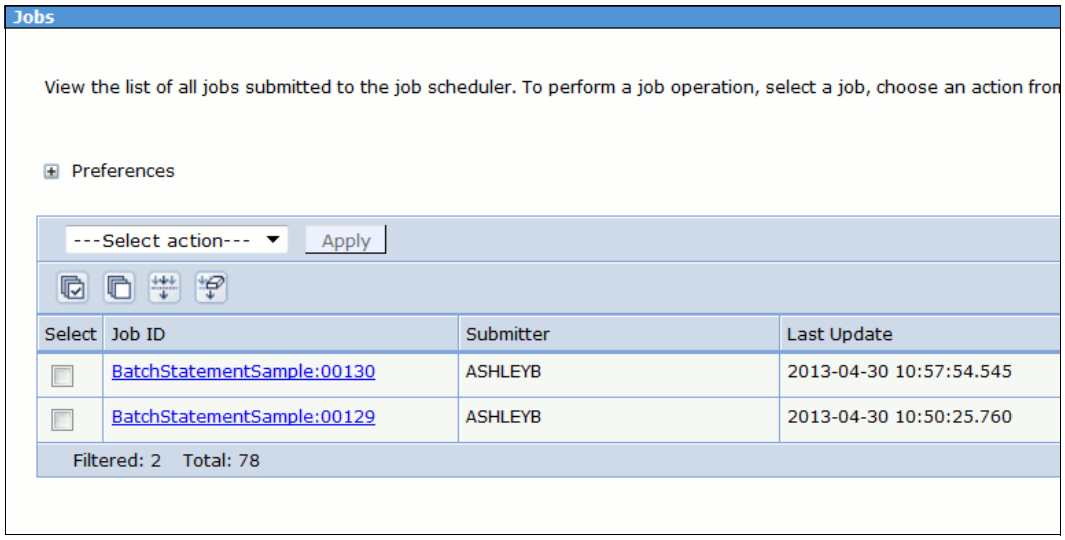

Figure 5-28 Example list of jobs from the Job Management Console

9. Click the Job ID link for the job you want to look at. In our example, the submitted job ID was BatchStatementSample:00130. The jobs log is displayed, as shown in Figure 5-29.

Job log

To save the log of job BatchStatementSample:00130 on the local file system, click **Download**.

```
CWLRB5671I: [04/30/13 10:57:54:040 GMT] Processing for job BatchStatementSample:00130 started.
CWLRB5832I: [04/30/13 10:57:54:070 GMT] Original XJCL
   1 : <?xml version="1.0" encoding="ASCII"?>
   2 : <job default-application-name="BatchStatementSample" name="BatchStatementSample" xmlns:xsi="h
   3 :     <job-type>Batch</job-type>
   4 :     <jndi-name>ejb/BatchStatementSampleBatchController</jndi-name>
   5 :     <step-scheduling-criteria>
   6 :         <scheduling-mode>sequential</scheduling-mode>
   7 :     </step-scheduling-criteria>
   8 :     <checkpoint-algorithm name="chkpt">
   9 :         <classname>com.ibm.wsspi.batch.checkpointalgorithms.recordbased</classname>
  10 :         <props>
  11 :             <prop name="recordcount" value="30"></prop>
  12 :             <prop name="TransactionTimeOut" value="5"></prop>
  13 :         </props>
  14 :     </checkpoint-algorithm>
  15 :     <results-algorithms>
  16 :         <results-algorithm name="jobsum">
  17 :             <classname>com.ibm.wsspi.batch.resultsalgorithms.jobsum</classname>
  18 :         </results-algorithm>
  19 :     </results-algorithms>
  20 :     <job-step name="AccountBalancePDFGenerator">
  21 :         <classname>com.ibm.itso.sample.AccountBalancePDFGenerator</classname>
  22 :         <checkpoint-algorithm-ref name="chkpt"></checkpoint-algorithm-ref>
  23 :         <batch-data-streams>
  24 :             <bds>
  25 :                 <logical-name>inputStream</logical-name>
  26 :                 <impl-class>com.ibm.cics.batch.bds.impl.VsamKsdsReaderImpl</impl-class>
  27 :                 <props>
  28 :                     <prop name="START" value="F0F0F0F0F0F0F0F0"></prop>
  29 :                     <prop name="KEYLENGTH" value="8"></prop>
  30 :                     <prop name="CICSFILE" value="SAMPIN"></prop>
  31 :                 </props>
  32 :             </bds>
  33 :         </batch-data-streams>
  34 :         <results-ref name="jobsum"></results-ref>
  35 :     </job-step>
  36 : </job>
CWLRB5833I: [04/30/13 10:57:54:087 GMT] Substituted XJCL
   1 : <?xml version="1.0" encoding="UTF-8"?>
```

Page: 1 of 2 Jump to page:

Figure 5-29 The Job log view in the Job Management Console

10. To see which Batch Container a job is dispatched too, we look for the message, `Job BatchSatementSample:00130 is dispatched to endpoint...` In our example, the Batch Container is IYK2ZMK3/DFHBATJS, which is the identifier for our CICS region's Batch Container (the name is defined by the CICS APPLID and JVMProfile name). An example of this job log message is shown in Figure 5-30.

```
CWLRB5684I: [04/30/13 10:57:54:112 GMT] Job BatchStatementSample:00130 is queued for execution
CWLRB5586I: [04/30/13 10:57:54:142 GMT] CWLRS6006I: Job class Default, Importance 8, Service Clas
CWLRB5586I: [04/30/13 10:57:54:144 GMT] CWLRS6007I: Job Arrival Time 4/30/13 10:57 AM, Goal Max C
CWLRB5586I: [04/30/13 10:57:54:145 GMT] CWLRS6021I: List of eligible endpoints to execute the job
CWLRB5586I: [04/30/13 10:57:54:146 GMT] CWLRS6011I: APC is not active. GAP will make the endpoint
CWLRB5586I: [04/30/13 10:57:54:199 GMT] CWLRS6013I: GAP is dispatching job BatchStatementSample:0
CWLRB3090I: [04/30/13 10:57:54:274 GMT] Job BatchStatementSample:00130 is dispatched to endpoint
```

Figure 5-30 Message showing which Batch Container job is dispatched too

Example 5-2 shows some of the output from a successful run of the Batch statement sample application. The debug output from the jobstep shows that a record is read from the input file, the COBOL business logic is called and returns an updated record, then a PDF file is created. The line stating "`chkpt checkpoint taken`" shows that a checkpoint was taken on the tenth iteration of the batch job step; that is, after processing the 10th record. A similar line of output would have been written after the fifth record earlier in the joblog.

Following some more output, which is not shown in Example 5-2, we see indications that the jobstep has completed normally with a return code of 0.

Example 5-2 Extract from the job log for the example application

```
System.out: [05/01/13 15:01:38:502 GMT] DEBUG->jobid:
BatchStatementSample:00736/AccountBalancePDFGenerator:AccountBalancePDFGenerator:processJobStep
System.out: [05/01/13 15:01:38:502 GMT] DEBUG->jobid:
BatchStatementSample:00736/AccountBalancePDFGenerator:Read input data : 77278883Phil Tomkins
0034300535
System.out: [05/01/13 15:01:38:503 GMT] DEBUG->jobid:
BatchStatementSample:00736/AccountBalancePDFGenerator:Calling COBOL business logic
System.out: [05/01/13 15:01:38:503 GMT] DEBUG->jobid:
BatchStatementSample:00736/AccountBalancePDFGenerator:Updated account data : 77278883Phil
Tomkins 0036015561
System.out: [05/01/13 15:01:38:504 GMT] Starting to create pdf file /u/batchfp/pdfs/77278883.pdf
for accountNumber 77278883
System.out: [05/01/13 15:01:38:511 GMT] ... invoice pdf file created.
CWLRB5628I: [05/01/13 15:01:38:526 GMT] Step AccountBalancePDFGenerator: chkpt checkpoint taken
[iteration 10]
 ….....................
CWLRB5630I: [05/01/13 15:01:38:557 GMT] Step AccountBalancePDFGenerator completes normally:
ended normally
CWLRB2460I: [05/01/13 15:01:38:558 GMT]  [05/01/13 15:01:38:558 GMT]  Job
[BatchStatementSample:00736] Step [AccountBalancePDFGenerator] is in step breakdown.
CWLRB5606I: [05/01/13 15:01:38:558 GMT] Destroying job step: AccountBalancePDFGenerator
System.out: [05/01/13 15:01:38:559 GMT] DEBUG->jobid:
BatchStatementSample:00736/AccountBalancePDFGenerator:entering
AccountBalancePDFGenerator:destroyJobStep
System.out: [05/01/13 15:01:38:559 GMT] INFO->jobid:
BatchStatementSample:00736/AccountBalancePDFGenerator:AccountBalancePDFGenerator.destroyStep()-
Total Execution Time: 176
System.out: [05/01/13 15:01:38:560 GMT] DEBUG->jobid:
BatchStatementSample:00736/AccountBalancePDFGenerator:destroyJobStep
CWLRB5608I: [05/01/13 15:01:38:581 GMT] Job step AccountBalancePDFGenerator destroy completed
with rc: 0
CWLRB2360I: [05/01/13 15:01:38:582 GMT] Job [BatchStatementSample:00736] Step
[AccountBalancePDFGenerator] finished with return code 0
CWLRB5610I: [05/01/13 15:01:38:583 GMT] Firing AccountBalancePDFGenerator results algorithm
com.ibm.wsspi.batch.resultsalgorithms.jobsum: [RC 0] [jobRC 0]
CWLRB5624I: [05/01/13 15:01:38:591 GMT] Stopping step AccountBalancePDFGenerator chkpt
checkpoint.  User transaction status: STATUS_ACTIVE
CWLRB5602I: [05/01/13 15:01:38:594 GMT] Closing AccountBalancePDFGenerator batch data stream:
inputStream
CWLRB5604I: [05/01/13 15:01:38:595 GMT] Freeing AccountBalancePDFGenerator batch data stream:
inputStream
CWLRB5854I: [05/01/13 15:01:38:596 GMT] Job Step
[BatchStatementSample:00736,AccountBalancePDFGenerator]: Metric = clock  Value = 00:00:00:140
CWLRB5854I: [05/01/13 15:01:38:596 GMT] Job Step
[BatchStatementSample:00736,AccountBalancePDFGenerator]: Metric = retry  Value = 0
CWLRB2600I: [05/01/13 15:01:38:597 GMT]  [05/01/13 15:01:38:597 GMT]  Job
[BatchStatementSample:00736] Step [AccountBalancePDFGenerator] completed normally rc=0.
CWLRB3800I: [05/01/13 15:01:38:603 GMT] Job [BatchStatementSample:00736] ended normally.
```

11. The batch job creates a PDF file for each account that is processed. These PDF files are stored in a directory on z/FS. An example of a generated file is shown in Figure 5-31.

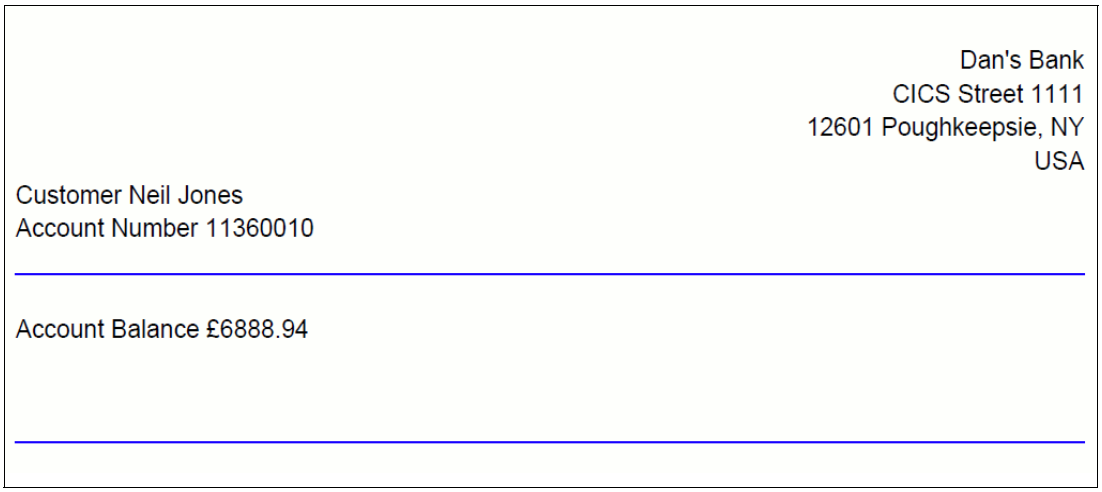

Figure 5-31 Example of PDF file created by the batch job

All the necessary material to create the sample described in this chapter can be downloaded from the Internet as described in Appendix A., “Additional material” on page 53.

5.6 Debugging a batch job step

When a job has been submitted, a job log is created for the job. This job log can be retrieved via the Job Management Console as described in “Submitting the xJCL to run the batch job” on page 37. This is the first place to look for errors that occurred during a batch job. If a Java exception occurs and the exception is caught and printed, it will be displayed in the job log for the job.

One way to catch exceptions is to surround the business logic with a try/catch statement and to print any exception that is caught, as shown in Example 5-3. Any exception that occurs inside the try block is caught, printed to SystemErr and then the exception is re-thrown so that the Batch Container knows something went wrong and can stop the job. You can then use the job log to see what the exception was and the line in the code that caused it.

Example 5-3 Using a try/catch statement to print Java exceptions

```
public int processJobStep() {
    try {
        .. perform business logic
    } catch (Exception e) {
        e.printStackTrace();
        throw new RuntimeException(“Unexpected error in batch loop”, e);
    }
}
```

An alternative to using the job log is to look directly at the SystemErr and SystemOut files created by the CICS JVMServer resource.

Applications can also be written to emit messages using a logger, as done by the BatchStatementSample application code. The benefit of writing debug messages using a logger, is that the logger takes care of the I/O associated with emitting the messages and can do so as a background task. The messages can be enabled or disabled depending on a property set in the xJCL for the batch job.In Example 5-4, a logger is created and used in the processJobStep() method to print out a message when the method is called. The xJCL for the batch job step contains a property called debug. If this property has the value true, then the debug message is created and can be seen in the jobs log.

See the sample code for the BatchStatementSample application for a complete example of how to use logging.

Example 5-4 Using a logger to put out debug messages

```
public class AccountsBalancePDFGenerator implements BatchJobStepInterface {

   protected BDFSWLogger logger;

   public void createJobStep() {
      logger = new BDSFWLogger(properties);
   }

   public int processJobStep() {
      if (logger.isDebugEnabled() {
         logger.debug("Entered processJobStep()");
      }
   }
}
```

The CICS API used to connect CICS Java programs to other CICS programs written in languages such as COBOL, ensures that errors that occur are reflected back to the calling program. For further details, see the Java Applications in CICS section of the CICS information center at this website:

http://publib.boulder.ibm.com/infocenter/cicsts/v5r1/index.jsp

5.7 Validation of the CICS connection to Batch Environment

In this section, we review how to confirm the CICS connection to the Batch Environment. The CICS Batch Container runs within a JVM server inside CICS. JVM servers write output to files on z/FS. Different files are created for output messages and error messages. These are known in Java as SystemOut and SystemErr. These files are created in the working directory for the Batch container, which by default is `<Batch install root>/batchfp/workdir`.

By default, the files have names beginning `<CICS region name>.<JVMServer name>`, for example:

```
SystemOut = IYK2ZMK3.DFHBATJS.dfhjvmout
SystemErr = IYK2ZMK3.DFHBATJS.dfhjvmerr
```

When the CICS Batch container starts, it will write messages to the BatchContainerLog log as shown in the following example of the CICS Batch container log (this will only happen if a log-level of INFO is specified in the `batchcontainer-config.xml` file).

```
ÝMon May 20 11:26:02 GMT 2013¨ 21 INFO Ýcom.ibm.cics.batch.impl.BatchStartup <init>¨ Batch Container Successfully Initiated
```

The batch container writes messages to the CICS log via the defined TDQUEUE: CBAT. When the CICS Batch container is started, it will write the following message to the CICS log:

```
CBCH CICSUSER <DATE/TIME> <JVMSERVER> Batch Container initiation requested
```

And, if the initiation of the CICS Batch Container is successful, it will follow this by the message:

```
CBCH CICSUSER <DATE/TIME> <JVMSERVER> DFHBATJS Batch Container initiation
completed
```

If the systems required by the CICS Batch container are unavailable, such as the JVMServer or the DB2 connection, then it will wait for these to become available. This will be shown in the CICS log with the message:

```
CBCH CICSUSER <DATE/TIME> <JVMSERVER> Batch Container waiting for JVM server
```

6

Extensions to the base example

In this chapter, we look at three extensions to the sample batch application:

- In the first extension, we show how JCL can be submitted to execute the sample batch application.
- In the second extension, we show how substitution properties can be passed into xJCL from JCL.
- In the third extension, we discuss how an external scheduler (such as Tivoli Workload Scheduler) can be used to schedule batch applications that run in the Batch Environment.

6.1 Invoking the BatchStatementSample application from JCL

The WebSphere Batch Environment on z/OS provides a program called WSGRID that can be called from JCL and invokes the job scheduler. This enables Batch Environment jobs to be invoked from JCL and to be used in combination with other traditional batch jobs. In this section of the chapter, we explain how we used WSGRID to invoke the BatchStatementSample application.

> **Note:** We do not cover the setup needed in the Batch Environment to use the WSGRID application. For further information on this setup, see the WebSphere Application Server 8.5 information center at this website:
>
> http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp

We follow these steps:

1. The WSGRID program can pass the location of XML JCL (xJCL) to the scheduler, or it can tell the scheduler the name of an xJCL document that is stored in the scheduler's job repository. We chose to store the xJCL for the BatchStatementSample application in the scheduler's job repository. From the Job Management Console shown in Figure 5-25 on page 38, select **Job Repository** → **Save a job**. In the Save a Job view that is shown, enter the following values, as shown in Figure 6-1.
   - Job name = `BatchStatementSample`
   - xJCL path = location where you stored the xJCL on your workstation

   Click **Save**. The xJCL is saved and can now be referred to by the name BatchStatementSample

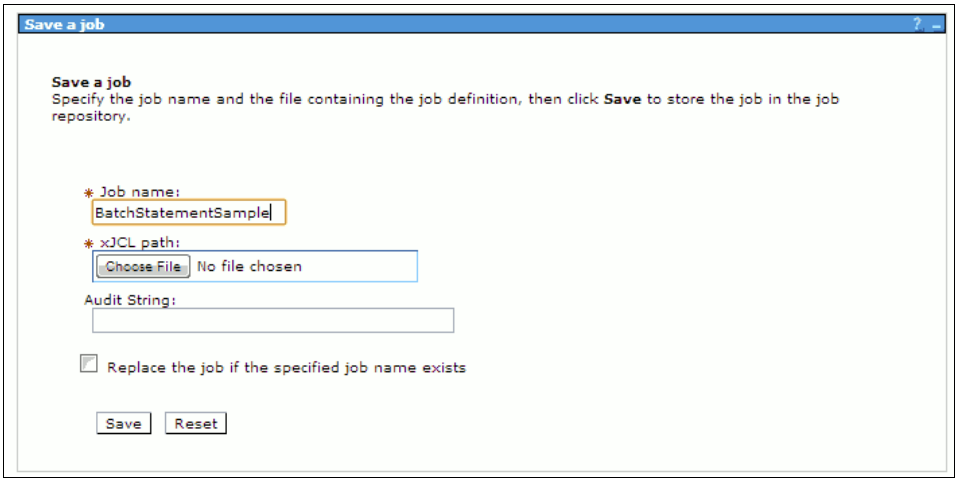

Figure 6-1 Saving the BatchStatementSample xJCL in the job repository

2. Create JCL to run the WSGRID program. Example 6-1 shows JCL which does this. The WSGRID program uses messaging to communicate with the job scheduler. The JCL tells the WSGRID program which queues to put the request message on and read the reply message from. The command repository-job=BatchStatementSample tells the job scheduler to read and submit the xJCL from the job repository. On return, the WSGRID program reflects the return code from the BatchStatementSample batch job so this return code can be used for checking in the JCL. The job log from the BatchStatementSample is also returned by the WSGRID program and written to the SYSPRINT DD card in the JCL.

Example 6-1 WSGRID JCL

```
//WSG8OCEL JOB (999,POK),MSGCLASS=H,
//  REGION=0M,NOTIFY=&SYSUID,JESLOG=NOSPIN
//RUN      EXEC PGM=WSGRID
//STEPLIB  DD DSN=SAMPLE.CG.LOAD,DISP=SHR
//         DD DSN=MQ701.SCSQLOAD,DISP=SHR
//         DD DSN=MQ701.SCSQAUTH,DISP=SHR
//*************************************************
//SYSPRINT DD SYSOUT=*
//*************************************************
//* WGCNTL DD SPECIFIES MQ INPUT/OUTPUT QUEUES
//* FOR THE WEBSPHERE Batch Environment JOB SCHEDULER.
//*************************************************
//WGCNTL   DD *
queue-manager-name=MQSZ
scheduler-input-queue=WASCG8IQ
scheduler-output-queue=WASCG8OQ
debug=true
/*
//*************************************************
//* WGJOB DD SPECIFIES THE XJCL JOB TO SUBMIT
//* TO THE WEBSPHERE Batch Environment JOB SCHEDULER.
//*************************************************
//WGJOB  DD *
repository-job=BatchStatementSample
/*
```

6.2 Passing substitution properties into xJCL from JCL

A useful addition to xJCL, is the ability to pass in substitution properties. In this example, we change the name of the CICS file from which the BatchStatementSample application reads its input. We make the name a substitution property with a default value of SAMPIN, so that the behavior when the xJCL is submitted is as before. We then show how a value can be passed from JCL into the xJCL, that overrides the CICS file name.

1. Using the xJCL editor in RAD, open the BatchStatementSample.xml file from the xJCL folder of the CreateStatements project. Highlight **Job (BatchStatementSample)** as shown in Figure 6-2 and click **Add**.

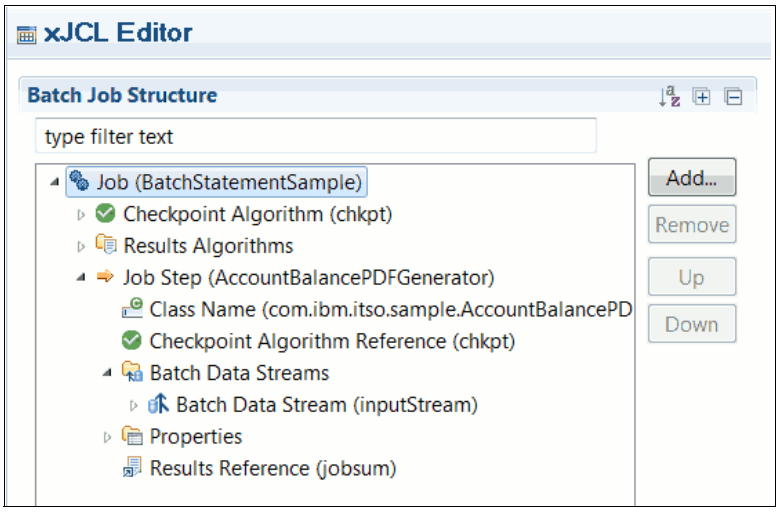

Figure 6-2 Adding a substitution property to xJCL through the xJCL editor

2. The Add Item dialog box is displayed, as shown in Figure 6-3. Highlight **Substitution Properties** and click **Ok**.

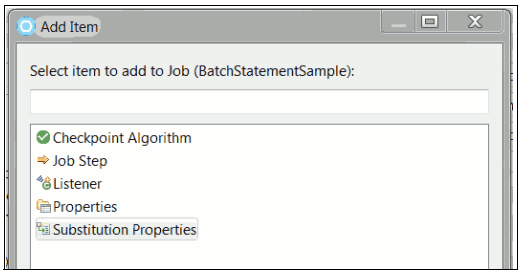

Figure 6-3 Add Item dialog box

3. The Substitution Properties dialog box is displayed, as shown in Figure 6-4. Click **Add** to add a new property. Give the property the following values
   - Name = CICS_FILENAME
   - Value = SAMPIN

   Click **Finish** to add the property.

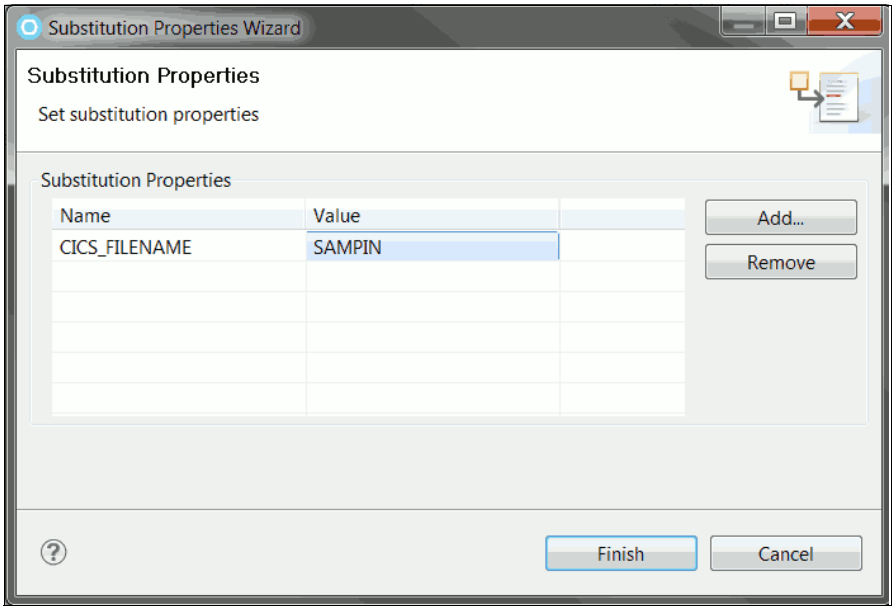

Figure 6-4 Default value for substitution property CICS_FILENAME is SAMPIN

4. Back in the xJCL editor, expand **Job (BatchStatementSample)** → **Job Step (AccountBalancePDFGenerator)** → **Batch Data Streams** → **Batch Data Stream (inputStream)** → **Properties** → **Property (CICSFILE)**.

   On the right hand side of the view, click **Choose Variable...** to change the Value of the CICSFILE property to the new substitution property. Select ${CICS_FILENAME} and click **Ok**. The property should look like Figure 6-5. Save the changes to the file.

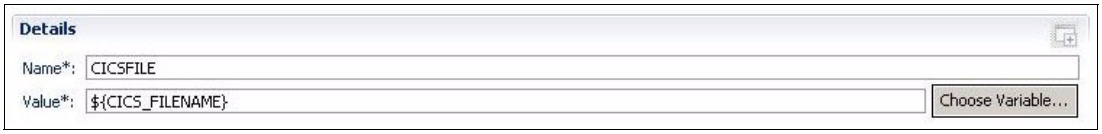

Figure 6-5 The CICSFILE property is set to use the substitution property CICS_FILENAME

5. Upload the new version of the xJCL into the job repository as shown in 6.1, “Invoking the BatchStatementSample application from JCL” on page 48. At this point, the xJCL should function as before. When submitted, the default value for the substitution property is used as no other value is specified.
6. Modify the WSGRID JCL as shown in Example 6-1 on page 49, to include the DD card as shown in Example 6. If you were to submit this JCL, the batch job in CICS would fail with a FileNotFoundException, unless you define SAMPIN2 as a CICS file, but this demonstrates that the substitution property is passed from JCL, via xJCL into the Batch Container in CICS, where it is used by the BatchStatementSample program.

Example 6-2 Passing xJCL substitution properties from JCL

```
//WGSUBS   DD *
substitution-prop.CICS_FILENAME=SAMPIN2
/*
```

6.3 Scheduling the BatchStatementSample from an external scheduler

Often, traditional batch jobs are scheduled using an external scheduler such as Tivoli Workload Scheduler (TWS). TWS can manage complex relationships between batch jobs, ensuring that the relevant job steps have completed before submitting further batch jobs. In a large organization which could be running thousands of batch jobs a day, a scheduling tool becomes very important.

As demonstrated in 6.1, “Invoking the BatchStatementSample application from JCL” on page 48, JCL can be created that when submitted, tells the job scheduler to submit a job. This is nothing in the JCL to differentiate it from any other JCL that TWS can submit and control. Therefore, Batch Environment jobs can be managed by an external scheduler through the use of standard JCL.

6.4 Summary

We have discussed the CICS TS Feature Pack for Modern Batch, which can be used to run batch jobs in a Java Virtual Machine in CICS. We have looked at when it may be appropriate to use this technology, discussed some of the design considerations to take into account, and walked through an example that enables a batch job to be driven from JCL and run in the CICS JVM.

For further information, see the following websites:

- CICS TS Feature Pack for Modern Batch Information Center:

  http://pic.dhe.ibm.com/infocenter/cicsts/v5r1/index.jsp

- WebSphere Batch Environment V8.5 Information Center:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp

A

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 to this website:

ftp://www.redbooks.ibm.com/redbooks/SG247991

Alternatively, you can go to the IBM Redbooks website:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247991.

Using the Web material

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

File name	*Description*
CICS_sample.zip	CICS sample as described in Chapter 5., “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/Linux

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 publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *Batch Modernization on z/OS,* SG24-7779
- *Java Application Development for CICS,* SG24-5275

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

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- WebSphere Batch Environment V8.5 Information Center

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp
- CICS Information Center

  http://pic.dhe.ibm.com/infocenter/cicsts/v5r1/index.jsp
- z/OS Java Technology Edition

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

  http://www-03.ibm.com/systems/z/os/zos/tools/java/
- Enterprise COBOL for z/OS Programming Guide

  http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp?topic=%2Fcom.ibm.entcobol.doc_4.1%2FPGandLR%2Fpgtitlemvs.htm

Help from IBM

IBM Support and downloads:

ibm.com/support

IBM Global Services:

ibm.com/services

(0.1”spine)
0.1”<->0.169”
53<->89 pages

New Ways of Running Batch Applications on z/OS

Volume 1 CICS Transaction Server

Technology overview

Application design considerations

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. A large portion of the workload on IBM z/OS systems is processed in batch mode.

This IBM Redbooks publication is the first volume in a series of four in which we specifically address 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 the latest support in CICS to run batch tasks.

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

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

SG24-7991-00 ISBN 0738437948