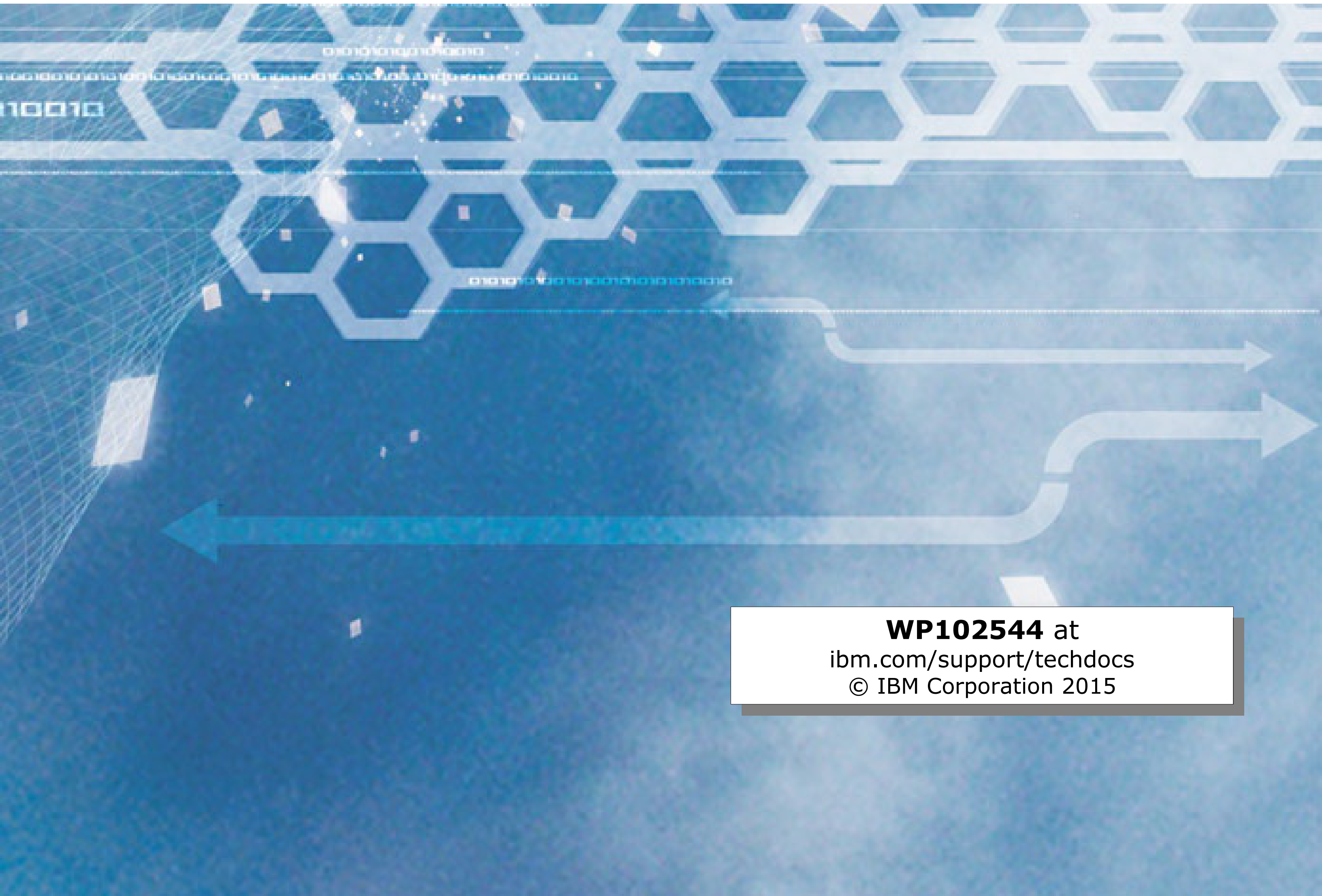
WebSphere Application Server for z/OS

Liberty Java Batch Step-by-Step Implementation Guide

Version Date: November 14, 2018

See "Document Change History" on page 84 for a description of the changes in this version of the document



WP102544 at
ibm.com/support/techdocs
© IBM Corporation 2015

Many thanks to **David Follis**, **Scott Kurz**, **Kaushik Mukherjee**, **Rob Alderman**, **Evan Jennings**, **Doug Macintosh**, and **Jeff Dutton** of the IBM WebSphere Application Server development and test organization.

Also: **Carl Farkas** and **Kevin Senior** of IBM France, and IBM UK, respectively.

Table of Contents

Introduction

Batch and Java batch

Batch processing has been a staple of information technology from the very early days. For example – batch jobs that create month-end statements, or batch jobs that calculate interest owed or fees on accounts. Batch jobs tend to operate against a large set of data, and unlike online processing – where each request completes very quickly – batch jobs are started and run for a while. For many years batch jobs were written in COBOL or Assembler.

Java batch is the same, except the language used is Java. That means a Java runtime environment is needed. But the *concept* of Java batch is the same as batch from before.

Java batch and z/OS

z/OS is a very good platform on which to run Java batch for several reasons:

- Batch data tends to reside on z/OS, and locating batch processing close to the data has the potential to reduce delays caused by network access.

  Batch programs are iterative by nature, with the batch program looping through the data for the job. Each call to the data – each read and write operation – involves latency. Reducing the *per call* latency is critical in batch processing because it tends to be *additive* across the life of the batch job. Less time per call leads to less time for the job as a whole.
- Java on z/OS is eligible for offload to speciality engines.

  This helps reduce the cost of running Java on z/OS. Speciality engines have a lower acquisition cost than general processors (GP), and processing cycles that run on the specialty engines do not count towards software license charges that are based on GP cycles.

Can Java perform as well as compiled COBOL?

A lot depends on the nature of the batch program, but the short answer is “very likely.” Here's why we say that:

- The Java Just-in-Time (JIT) compiler will watch for and compile Java class files it sees being invoked frequently. Batch is by its nature iterative, which means the Java batch class files will be JIT'd fairly quickly. Once JIT'd, execution is performed using compiled modules.
- The Java JIT compiles its code based on knowledge of the latest features of the underlying processor on the System z platform. This allows the JIT to compile code optimized for instructions available on the hardware. Depending on when the COBOL was last compiled, the COBOL batch job may not have the same efficiencies.

JSR 352 specification

Back in 2011, representatives from several companies formed a group to come up with a standard programming specification for Java batch. That project was assigned the number “JSR 352,” and it is that number that is used to refer to open standard Java batch.

The result of that group's efforts is the JSR 352 specification for Java batch, the first release of which was dated 24-May-2013. That specifications webpage is:

`https://jcp.org/en/jsr/detail?id=352`

For more on this specification, see “Overview of the JSR 352 specification” on page 7.

IBM implementation of JSR 352

The specification spells out the programming *interfaces*, but the actual implementation is left up to the vendors that wish to provide the function. IBM is implementing JSR 352, and IBM's implementation of it is the focus of this document.

For more on this, see "Overview of the IBM implementation of JSR 352" on page 10.

Minimum requirements

To use the JSR 352 support you must be at IBM Liberty Profile 8.5.5.6 or above. This applies to any operating system platform on which Liberty Profile is supported, including z/OS.

What about the WebSphere Java Batch function, formerly known as Compute Grid?

It still exists, and is part of the WebSphere Application Server V8.5 product.

JSR 352 and WebSphere Java Batch (formerly "Compute Grid") have different programming models, so a batch program written for one can not run unchanged in the other. There are programming approaches to wrapper the batch logic in a way to be usable in either environment. It would involve changing the wrapper and repackaging the application. But it is possible. How that's done is outside the scope of this document.

Layout of this document

This document is designed to guide the reader through understanding what IBM's JSR 352 Java Batch function is, how to set it up, and how to use it:

Section	*Page*
Overview In this section we set context and provide a framework of understanding what the IBM JSR-352 implementation is and how it works.	7
Setup and Validation Before you can use IBM's JSR-352 Java Batch runtime, you need to install, setup and validate it. We provide that information here.	13
Review of the JSR 352 Sample Applications In this section we take a look at the two sample applications used for validation – SleepyBatchlet and BonusPayout – to get a sense for how a JSR 352 Java Batch application is constructed.	51
Miscellaneous Information Every document has information that is interesting and relevant, but that does not *quite* fit inline with the topic of a section. We hold this information here, under "miscellaneous."	64

Overview

The intent of this section is to provide you with an understanding of JSR 352 and how IBM is implementing it. With that framework understood, the details in the following sections will make more sense.

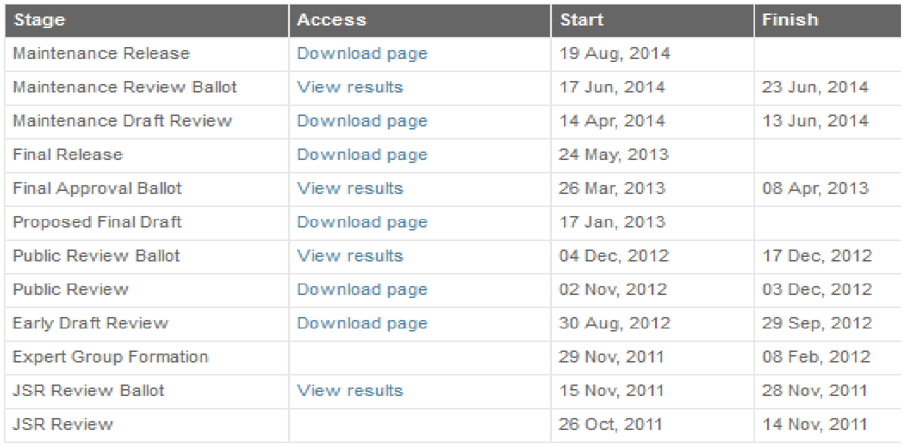
Overview of the JSR 352 specification

JSR 352 is the open community's standard for Java batch. Development of the standard began in 2011, and the initial specification was released in 2013. IBM was the specification lead, with participation from other companies, as is common with open standard development.

As noted earlier, the JSR 352 specification web page is found here:

https://jcp.org/en/jsr/detail?id=352

That page shows the progression of activity leading towards release, and further work on updates to the specification:



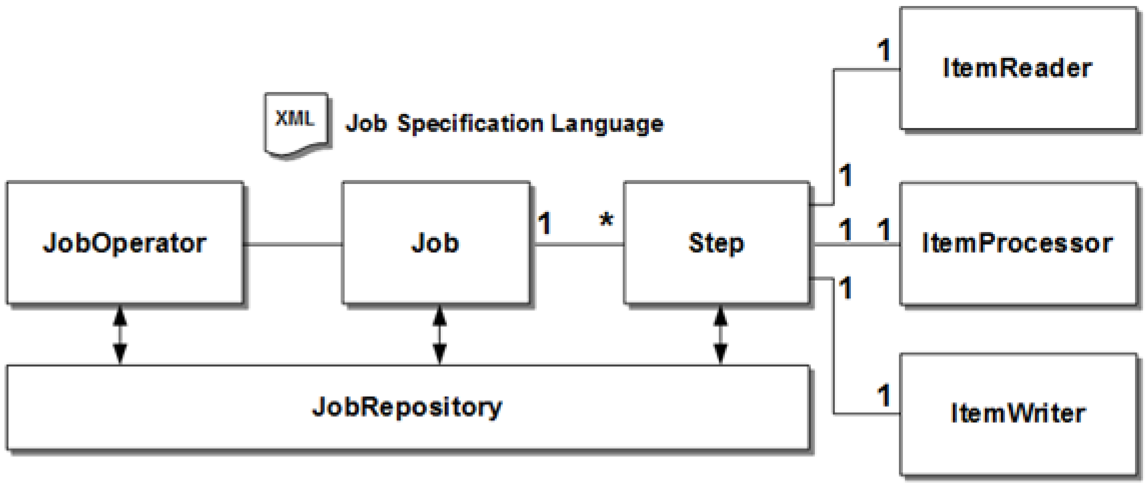
Stage	Access	Start	Finish
Maintenance Release	Download page	19 Aug, 2014	
Maintenance Review Ballot	View results	17 Jun, 2014	23 Jun, 2014
Maintenance Draft Review	Download page	14 Apr, 2014	13 Jun, 2014
Final Release	Download page	24 May, 2013	
Final Approval Ballot	View results	26 Mar, 2013	08 Apr, 2013
Proposed Final Draft	Download page	17 Jan, 2013	
Public Review Ballot	View results	04 Dec, 2012	17 Dec, 2012
Public Review	Download page	02 Nov, 2012	03 Dec, 2012
Early Draft Review	Download page	30 Aug, 2012	29 Sep, 2012
Expert Group Formation		29 Nov, 2011	08 Feb, 2012
JSR Review Ballot	View results	15 Nov, 2011	28 Nov, 2011
JSR Review		26 Oct, 2011	14 Nov, 2011

You can download the specification PDF from that page.

Like most active open standard specifications, the work is ongoing. The specification page is the best source of information on the latest status for the specification.

Architecture diagram and terminology

The specification document provides a diagram used as a backdrop for the discussion of the components of the specification. That diagram looks like this:



Let's review the pieces of that diagram and start to paint the picture of what a JSR 352 environment consists of:

- ***Job*** – this is the logical entity that wrappers the batch process. The details of the job are spelled out in the *Job Specification Language* (see below) which is an XML file. The *concept* of a JSR 352 job is exactly the same as a JCL batch job.
- ***JobOperator*** – this is the operational interface through which you submit and monitor the jobs.
- ***Job Specification Language*** – an XML document that provides the details of the Java batch job to be run. In concept this plays the same role as traditional JCL has played for decades. The difference is the document is coded using XML.
- ***JobRepository*** – this is where the details of jobs, either currently running or having run in the past, is maintained. The specification does not spell out the details of this, but in the case of IBM JSR 352 this is a relational database.
- ***Step*** – this is the heart of the batch processing. The JSR 352 specification document provides a very nice description of this:

  > A Step contains all of the information necessary to define and control the actual batch processing. This is a necessarily vague description because the contents of any given Step are at the discretion of the developer writing it. A Step can be as simple or complex as the developer desires. A simple Step might load data from a file into the database, requiring little or no code, depending upon the implementations used. A more complex Step may have complicated business rules that are applied as part of the processing.

  A job may contain one or more steps, depending on the design of the batch job.

The next three items are related to the job step on a one-to-one basis. That means a given job step will define one of each, but *not* multiple of any given one:

- ***ItemReader*** – the ItemReader is what retrieves data for the job step to operate on. An “item” of data is whatever the developer determines is appropriate, based on the batch processing that needs doing. That may be a row from a database table, or it may be data from several different sources merged into a data “item” for processing.

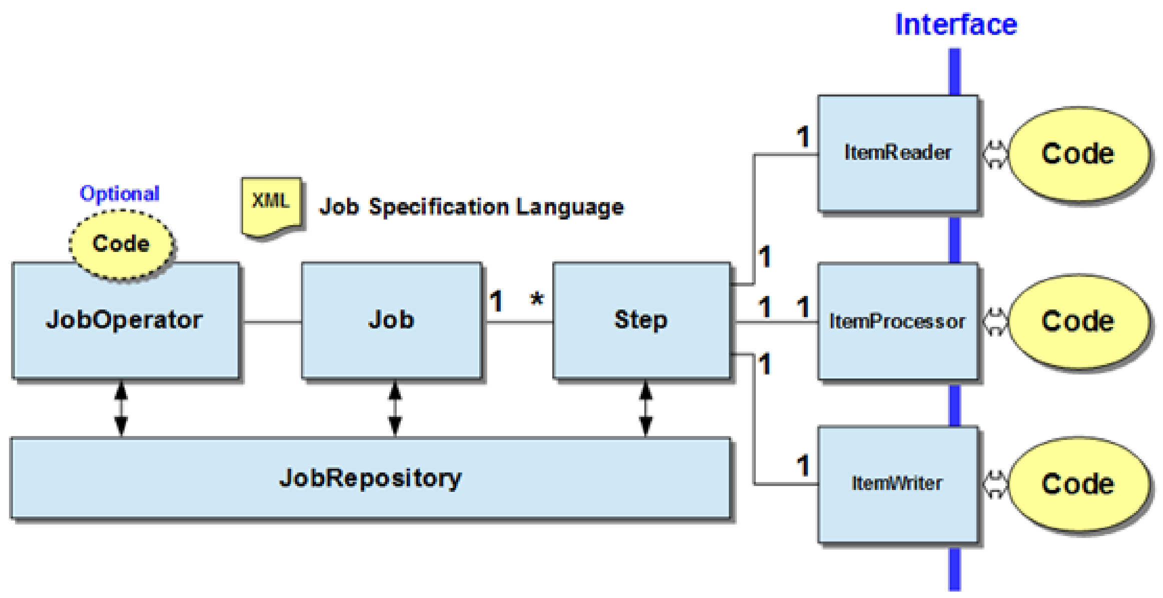
  The ItemReader signals when the data it is retrieving has been exhausted. The Step invokes the ItemReader (and ItemProcessor[1]) as long as the ItemReader indicates there's still data available.
- ***ItemProcessor*** – the ItemProcessor is what performs the business logic upon the data item that was provided by the ItemReader. Whatever the business requires – calculate interest, format a month-end statement – is done here.

  > This is where a multi-step job comes into the picture. Your batch processing may involve several phases (or steps) of processing upon the data. The first step may read and process the data to an intermediate state; the second step then sweeps through and does final processing. Whether your job has one step or a dozen is up to you.
- ***ItemWriter*** – the ItemWriter is called by the step to write the output of the ItemProcessor. But here's the key – the ItemWriter is called when the specified *chunk* (another word for “collection,” or “group”) of output data is gathered up. The chunk of data is passed to the ItemWriter, which writes it out and commits the data.

  The concept of chunking is the same as checkpoint processing. The value for the ItemWriter chunking is specified in the Job Specification Language (XML) file, and it represents the interval at which commits or rollbacks will take place.

If we return to the architecture diagram from earlier and update it to indicate what portion of the architecture is implemented by vendors, and what portion is implemented by you, the picture looks like this:

1 ItemWriter is deliberately left out of this for the moment. See the ItemWriter description that follows.



Those boxes in light blue are the responsibility of the vendor implementing the JSR 352 environment. You, the Java developer, supply four things:

- The Java to implement the ItemReader behind the ItemReader interface
- The Java to implement the ItemProcessor behind the ItemProcessor interface
- The Java to implement the ItemWriter … again, behind the interface, and
- The XML to describe the job and steps in the Job Specification Language file, according to the specification guidelines for the format and syntax of that file.

What about the dotted oval marked “optional?” That depends on whether you want to write your own function that uses the JobOperator API, or if you want to rely on function provided by a vendor. For example, the SleepyBatchlet sample is packaged with a servlet for job submission and control[2].

However, you could use a vendor-supplied function to submit jobs and manage their execution. IBM supplies a REST interface function for this purpose, which means you would *not* need to provide code for that portion[3].

The next question is this – *what does this environment look like as supplied by IBM, and how do I operate this thing?* Answering those questions is the purpose of this document.

JSR 352 step types

The specification defines two types of steps: *chunk* and *batchlet*:

- ***Chunk*** – this is batch processing step model most people think of when they think of batch: an iterative loop through a set of data, with periodic checkpoint and commits. The discussion above about ItemReaders, ItemProcessors and ItemWriters applies to this step model.
- ***Batchlet*** – this is for batch processing that is *not* item-oriented. For example, a batch step that performs a large file transfer operation. Or a batch step that performs a lengthy set of data calculations.

2 See “The SleepyBatchlet sample” on page 51 for more on the internals of that sample batch application.
3 See “Configure batchManagement-1.0 support” on page 23 for more.

Another use-case for this step model is as a way to run existing Java `main()` programs[4] in the JSR 352 runtime with a minimum of programming changes.

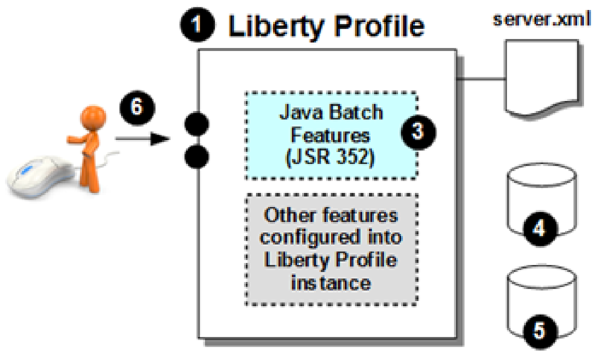
Depending on the nature of the batch processing being done in the Java `main()` program, you may wish to one day re-engineer it to take advantage of the Chunk model. But as a short term transition, running as a Batchlet provides a way to re-use assets relatively easily.

Which you choose to implement is going to depend on the nature of the batch step you wish to process. Iterative, transactionally-oriented processes are better suited to Chunk. Batchlet is what you would use for other tasks.

Overview of the IBM implementation of JSR 352

The JSR 352 specification defines the *interfaces* to batch processing functionality. The code that implements the function behind the interfaces is left up to the vendors who choose to provide a JSR 352 environment.

IBM's initial implementation of JSR 352 is in Liberty Profile, with the JSR 352 implemented as *features* of Liberty. The following illustrates this. See the notes that follow.


1 Liberty Profile
server.xml
Java Batch Features (JSR 352)
3
Other features configured into Liberty Profile instance
6
4
5
JSR 352 persistence (JobRepository)
JSR 352 batch applications

```
<featureManager>
  <feature>batch-1.0</feature>
  <feature>batchManagement-1.0</feature>
</featureManager>
```

Notes:

1. This is a Liberty Profile server instance. It can be on z/OS or on any platform supported by IBM Liberty Profile. The Liberty Profile server instance is created in the usual way[5]; that is, there's nothing about JSR 352 that requires anything special in creating the server initially.
2. The JSR 352 Java batch features are enabled in this instance with potentially two updates to the `<featureManager>` section of the `server.xml` – `batch-1.0` and `batchManagement-1.0`.

   The two elements[6] do the following:

   - `batch-1.0` – this enables the core JSR 352 support
   - `batchManagement-1.0` – this enables job logging, a REST interface for remote job submission, and the batchManager command-line utility[7]

   For more on the REST interface and the batchManager utility, see "Configure batchManagement-1.0 support" starting on page 23.
3. Updates to the `server.xml` (❷) is used by Liberty Profile to load the JSR 352 Java batch features.
4. The JSR 352 specifically calls for a JobRepository, which is a place to hold information about the jobs that are running and those that have run in the past. This is a relational data store. Its actual implementation depends on how robust you need it to be:

4 These programs are typically invoked with a JVM launcher function such as JZOS. JZOS is both a JVM launcher and a set of class and native libraries to access z/OS functions. If you use the JZOS libraries and find them useful, you can still use them with the IBM JSR 352 runtime on z/OS. It's just a matter of getting the JZOS libraries on the `CLASSPATH` and `LIBPATH` of the JSR 352 runtime.

5 For Liberty Profile on z/OS, see `ibm.com/support/techdocs` and search on WP102110.

6 If you code *just* the `batchManagement-1.0` element, Liberty Profile knows that `batch-1.0` is implied and will load that. The reverse is *not* true: coding just `batch-1.0` will *not* result in `batchManagement-1.0` being loaded.

7 If just `batch-1.0` is coded, the JobOperator API is available but requires some function *you* provide to use the API to submit the job. This is what the SleepyBatchlet sample does with the servlet packaged with the sample.

Development	For development, where the focus is on the application and not the infrastructure, an in-memory copy of this persistence store is possible. That in-memory copy goes away when you stop and restart Liberty Profile, so it's not really "persistent". For development that's likely to be acceptable. This is the simplest configuration model. But if you need it to survive restarts of your Liberty Profile server, then one of the other options is available.
System Testing	Here you may want the data in the persistence store to … well, *persist*. For example, information on the last checkpoint taken is needed to test recovery after a system outage. To accomplish this you create[8] a set of relational tables (either in a file-based function like Derby) or a relational product like DB2[9], and you configure the access to that with XML updates to the `server.xml`.
Production	In production you will definitely need a robust persistence store. This will likely be in a true relational product such as DB2. On z/OS, that may be DB2 in a data sharing mode for even more robustness.

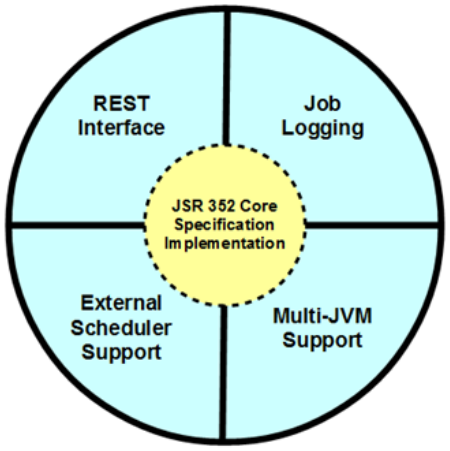
5. Your batch programs are deployed into the Liberty Profile. This is standard Liberty Profile deployment: the applications are either placed in the `/dropins` directory, where they are detected dynamically by Liberty Profile, or they are pointed to from the `server.xml`. Either way, there is nothing different about *deploying* JSR 352 batch applications into the Liberty Profile server.
6. With the feature elements added (❷) and the JSR 352 function enabled (❸), you are now able to invoke the batch application through the listener ports of the the Liberty Profile server instance.

The physical runtime is not that difficult to establish. The creation of a Liberty Profile server is relatively easy; editing the `server.xml` is just that – editing; and deploying the batch application is a standard Liberty Profile practice. Creating the persistent store tables (or using the in-memory option) is a relatively easy and is covered under "Create JSR 352 job respository data store" on page 14.

IBM operational extensions to the JSR 352 standard

The JSR 352 standard is largely a *programming interface* standard. It does not, at present, spell out *operational* features.

IBM's enhancements in this area are summarized by this picture:


REST Interface
Job Logging
JSR 352 Core Specification Implementation
External Scheduler Support
Multi-JVM Support

Each is briefly described next, with pointers to sections in this document where each topic is discussed in more detail:

8 How this is done is covered under "Create JSR 352 job respository data store" on page 14.
9 The Java batch function is tested for use on the following database product versions: DB2 LUW v10.1, DB2 z/OS v10.1, Oracle 11g, and Apache Derby.

- ***REST Interface*** – this is an IBM job management implementation that uses the JobOperator interface of JSR352 to submit, monitor and manage the batch jobs that run in the JSR 352 environment. Configuring and using this is described under "Configure batchManagement-1.0 support" on page 23.
- ***Job Logging*** – in the absence of a function to handle batch job logging separately, the results would show up in the standard server output. If the server is running many batch jobs, this becomes a challenge to keep track of things. IBM's job logging function creates a separate log file for each job execution, separate from the server output. This is part of the `batchManagement-1.0` function, and is described under "Configure batchManagement-1.0 support" on page 23.
- ***External Scheduler Support*** – enterprise schedulers, such as IBM's Tivoli Workload Scheduler[10], play a significant role in enterprise batch operations. The ability to schedule the JSR 352 Java batch execution with an enterprise scheduler is important. We describe this under "External scheduler support" on page 36.
- ***Multi-JVM Support*** – this allows jobs to be *submitted* through the JobOperator interface of one server, and *run* in another JVM with JSR 352 support configured. This implements the dispatcher and executor model of batch processing. We illustrate Multi-JVM support starting on page 42.

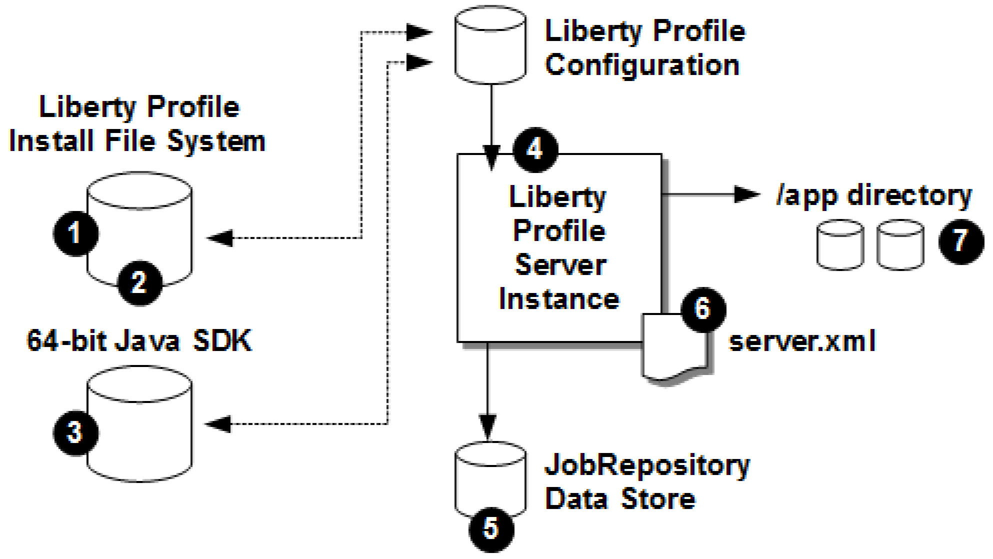
10 Or CA-7, or Control-M.

Setup and Validation

In this section we will provide a step-by-step guide to setting up and validating the JSR 352 runtime environment.

Installation overview

The following diagram illustrates the key steps to setup and validation. Notes follow and correspond to the numbered circles:


Liberty Profile Configuration
Liberty Profile Install File System
1
2
64-bit Java SDK
3
4
Liberty Profile Server Instance
/app directory
7
6
server.xml
JobRepository Data Store
5

Notes:

1. Liberty Profile at the level 8.5.5.6 or above installed and available. Installation of Liberty Profile is accomplished using IBM Installation Manager (IM).

   For Installation Manager z/OS, see WP102014 at `ibm.com/support/techdocs`.

2. The Liberty features necessary for using the JSR 352 function are installed. See "Installing the JSR 352 features into Liberty Profile" on page 64 for more on this.

3. A valid 64-bit Java SDK available for use by Liberty Profile. The JSR 352 function will work with either Java 7 or Java 8.

   For z/OS, see `http://www.ibm.com/systems/z/os/zos/tools/java/index.html`

4. A Liberty Profile server instance. This is accomplished with the supplied `server create` function as has been the case since the beginning of Liberty Profile.

   For Liberty Profile z/OS, see WP102110 at `ibm.com/support/techdocs` … specifically, the “Liberty Profile z/OS Quick Start Guide” provided there.

5. A JSR 352 job repository persistence data store established. Alternatives here range from very simple *(in-memory, no configuration)* to more robust (in a database product such as IBM DB2).

   See “Create JSR 352 job respository data store” on page 14.

6. The `server.xml` file for the Liberty Profile server instance configured to enable the JSR 352 function as well as point to the Java batch job repository data store.

   See “Configure JSR 352 batch-1.0 support in server.xml” on page 19.

7. Java batch applications in an accessible directory. For initial setup and validation the supplied samples work well. This may be the `/dropins` directory for the Liberty Profile server, or a directory pointed to with `<application>` or `<webApplication>` elements in the `server.xml`.

Setup step-by-step

Create Liberty Profile server instance

The *creation* of a Liberty Profile server in which JSR 352 will operate is really no different than for any other use. The `server create` function is used to create the server instance.

For Liberty Profile z/OS, the server instance may be started as either a UNIX process or a z/OS started task. The core JSR 352 support imposes no requirement on the z/OS operational model chosen.

The WP102110 Techdoc provides a "Quick Start Guide" to provide an overview and step-by-step for the creation of a server instance on z/OS. That Techdoc can be found here:

`http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102110`

Do the following:

- ☐ Use the WP102110 Quick Start Guide (or your own knowledge) and create a *basic* Liberty Profile server instance.

  Up through server create and, if you wish, the steps to start it as a z/OS started task. For now[11] you do *not* need the other things shown in that document: security, Angel process, or WLM or z/OS MODIFY. Just a basic server that starts with the default configuration.

- ☐ Update the `server.xml` with the following:
  - ○ Determine if the `httpPort=9080` default value is acceptable for your environment. If not, change it to a value that is acceptable.
  - ○ Add the following element:

    ```
    <httpEndpoint id="defaultHttpEndpoint"
       host="*"
       httpPort="9080"
       httpsPort="9443" />
    ```

  - ○ Remove the `jsp-2.2` feature:

    ```
    <featureManager>
        ~~<feature>jsp-2.2</feature>~~
    </featureManager>
    ```

  - ○ Add the `servlet-3.1` feature:

    ```
    <featureManager>
        <feature>servlet-3.1</feature>
    </featureManager>
    ```

  - ○ Save the file.

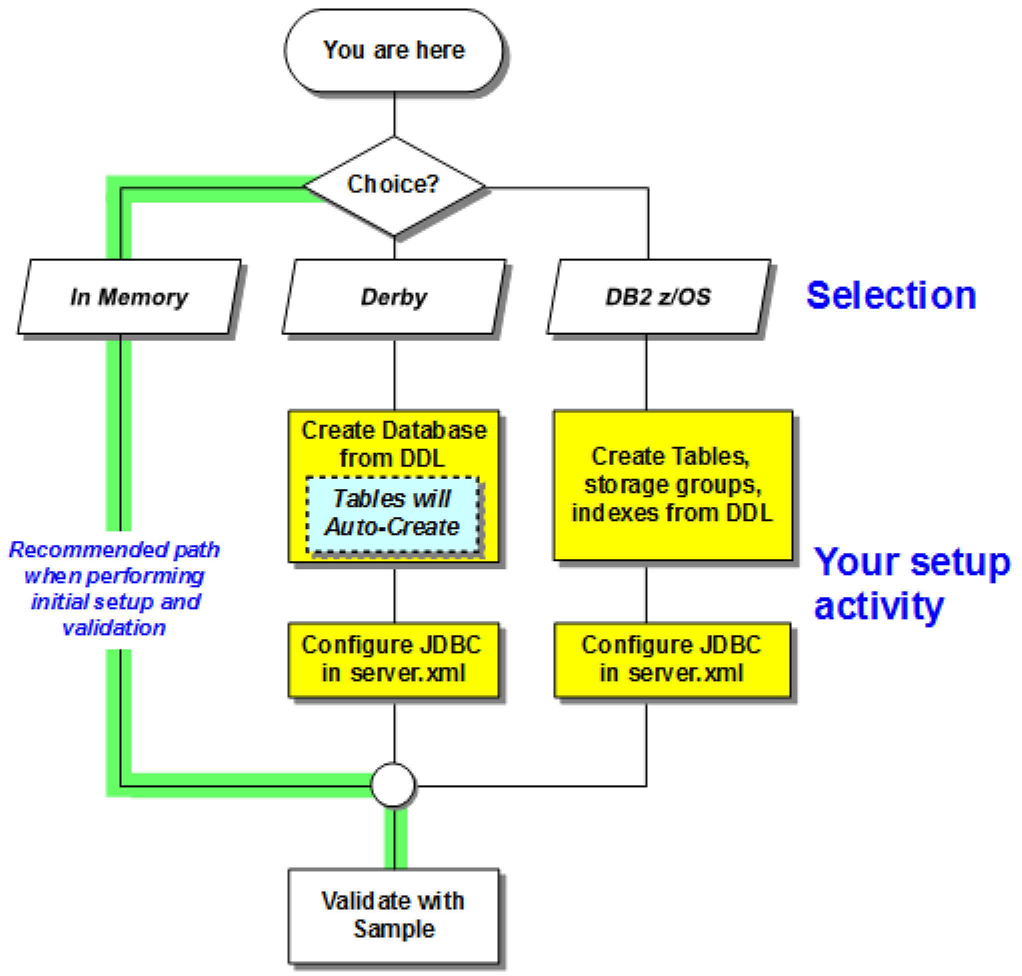
- ☐ Test to insure the server starts. Inspect the `messages.log` file to make sure the server starts without errors.

Create JSR 352 job respository data store

JSR 352 requires a JobRepository (data store) to maintain information about batch jobs. With IBM JSR 352 you have three options, from *very simple* (in memory, no setup) to *very robust* (database product, such as IBM DB2, which requires some setup).

The following picture illustrates the paths you may take in setting up your environment:

11 Later, when you use the `batchManagerZos` function (starting on page 39), that requires the Angel process because it uses WOLA. WOLA is an authorized service, and the Angel is what Liberty Profile z/OS uses to protect access to authorized services.


You are here
Choice?
In Memory
Derby
DB2 z/OS
Selection
Create Database from DDL
Tables will Auto-Create
Create Tables, storage groups, indexes from DDL
Your setup activity
Recommended path when performing initial setup and validation
Configure JDBC in server.xml
Configure JDBC in server.xml
Validate with Sample

Clearly the path highlighted in green is the easiest way to begin. That is the in-memory option and requires no setup of databases or tables, and requires no JDBC definitions.

Note: you are ***not*** locked in once you choose an JobRepository option. You can easily change your runtime environment to one of the other options. You simply configure for another option and you have a fresh set of tables to work with[12].

Choose one and go to the page indicated:

Format	*Description*	*Page*
In-memory	The simplest to begin with, though information does *not* persist between restarts of the Liberty Profile server. Good for initial validation and development, but not for system testing or production. ***We recommend this when first starting with IBM JSR-352***	16
Derby DB	Derby is an open-source file-based relational database function. Information persists between restarts of Liberty Profile server. Good for development and test but not for robust production.	16

12 The information in the old tables is *not* migrated to the new choice. Changing options implies starting with a new set of tables.

Format	*Description*	*Page*
Database Product	For example, IBM DB2[13]. This may be used for development, test or production.	18

In-memory

This is the easist because *you don't have to do anything*. If there's no configuration in place for the JSR 352 persistence, it assumes in-memory. Do the following:

- ☐ Go to "Configure JSR 352 batch-1.0 support in server.xml" on page 19.

Derby

Apache Derby[14] is an open source Java relational database system that is relatively easy to create and use. The database takes the form of a set of files. A program such as IBM's JSR 352 batch accesses a Derby database using standard JDBC.

The *tables* will be created automatically by the IBM Java Batch feature. But the *database* must be created manually by you ahead of time. That is what you will do next.

Do the following:

- ☐ Determine the location of the Java 64-bit SDK used by your Liberty Profile server instance[15]. Note the path to the `/bin` directory here:

- ☐ Determine if you have the Derby class files on your system, and if so where they are located. Files to search for: `derby.jar` and `derbytools.jar`.

  If you have WAS z/OS full-profile installed, it will be under:

  `/<mount>/derby/lib`

  Note the location here:

- ☐ If you do *not* have the Derby class files on your system, then do the following:
  - ○ Go to the Derby download site and pull the latest release to your workstation:
    `http://db.apache.org/derby/derby_downloads.html`
    (The "lib" distribution is all you need.)
  - ○ Unzip the downloaded file. You should see a set of JAR files, including `derby.jar` and `derbytools.jar`.
  - ○ Create a directory on your system to hold the Derby JAR files. A convenient place is to create a directory `/derby` under the `/resources` directory of your Liberty Profile instance configuration.
  - ○ Upload all the Derby JAR files in binary to the directory you created.
  - ○ Make sure the permissions on these files offer at least `READ` to the ID under which the Liberty Profile server will run.
  - ○ Note the location of the directory here:

13 The Java batch function is tested for use on the following database product versions: DB2 LUW v10.1, DB2 z/OS v10.1, Oracle 11g, and Apache Derby.
14 See https://db.apache.org/derby/
15 The command to create the Derby database involves invoking `java`. We need to make sure the `/bin` directory of the SDK is on your `PATH`. The Java used for this does not need to be the same as what Liberty Profile uses. If some other Java is already on your `PATH`, then you can skip this step.

- ☐ Determine where you want the Derby database to reside. It may reside at any location. A convenient place to locate it is under the `/resources` directory of the Liberty Profile instance configuration directory. Note the location here:
- ☐ Create a DDL file on your workstation – any name you wish – and populate it with the following:

```
CONNECT 'jdbc:derby:BATCHDB;create=true';
COMMIT WORK;
DISCONNECT;
```

- ☐ Save the file.
- ☐ Upload the file *in binary* to your system[16] and set UNIX permissions so the file is readable by the ID you will use to run the Derby create command.
- ☐ Open a Telnet session or OMVS with your host system.
- ☐ Make sure java is on your `PATH`. Enter the command `java` and see if it is recognized. If not, then enter this command:

```
export PATH=$PATH:<path to /bin of Java SDK>
```

- ☐ Change directories to the location where you wish the Derby database to be created[17], for example the `/resources` directory of the Liberty Profile configuration directory.
- ☐ Issue the following command *all on one line*:

```
java -Djava.ext.dirs=<path to Derby JAR>
    -Dij.protocol=jdbc:derby: org.apache.derby.tools.ij
                                  </<path and file of DDL>>
```

Where

- `<path to Derby JAR>` – is where you placed the `derby.jar` and `derbytools.jar` files.
- `<path and file of DDL>` – is where you placed the file you created with the three lines of DDL to create the database.

You should see the process read in the DDL file and execute the commands found there.

Some common problems at this point that may require your attention:

- Lack of `READ` to the Derby JAR files
- Lack of `READ` to the DDL file
- Lack of `WRITE` needed to create directory and files
- Typographical errors in paths … path and file names are case-sensitive

- ☐ You should see a directory created with the name `BATCHDB`[18]. You should also see a `derby.log` file in the directory where the database create command was entered.
- ☐ **Important:** `chown` the `derby.log` file and *recursively* `chown` the database directory so the ID under which the Liberty Server instance runs has ownership:

16 If z/OS, Derby expects the input file to be in ASCII. Uploading an ASCII file in *binary* preserves the ASCII file encoding.
17 By default, Derby will create the database starting from the path location where the command is entered.
18 That was the database name in the first line of the DDL file you created.

```
chown <ID> derby.log
chown -R <ID> <database directory>
```

The database is created, but the tables are not yet created. The IBM Java Batch feature will create those after you give it knowledge of this Derby database. This requires updates to the `server.xml`. That's covered under "JDBC definitions for JSR 352 job repository data store access" starting on page 19.

Database product (DB2 z/OS)

There are two ways this can be done:

Auto-create	If you configure JDBC access to DB2 z/OS and the tables do not pre-exist, the persistence function will *attempt* to auto-create them[19] [20]. This may or may not work, depending on the authority of the ID under which Liberty Profile runs. Further, your data administrator may not wish you to dynamically create tables, preferring instead that they be manually created. If you wish to auto-create, see "JDBC definitions for JSR 352 job repository data store access" starting on page 19, then go to the section for DB2 z/OS (page 20).
Pre-define	This is the more conventional way to define table structures in DB2 z/OS. This is what we'll cover below.

A shell script is supplied that will read the persistence definitions in the `server.xml` for a server and from that generate a file with the DDL statements needed. Then you work with your DBA to customize the DDL to your environment and create the tables.

To generate the DDL for DB2 z/OS, do the following:

- ☐ Stop your server if it is running.
- ☐ Open `server.xml` for edit, and insert and customize the JDBC updates found under "DB2 z/OS" on page 20.
- ☐ Add the following two features[21]:

```
<featureManager>
    <feature>servlet-3.1</feature>
    <feature>localConnector-1.0</feature>
    <feature>batch-1.0</feature>
</featureManager>
```

- ☐ Save the file.
- ☐ Make sure the DB2 to which the server connects is up and running
- ☐ Start the server.
- ☐ Open a Telnet session to your z/OS system and log in.
- ☐ Change directories to your server's directory where `server.xml` lives.
- ☐ Issue the following command to put the 64-bit Java SDK /bin directory on your path:

```
export PATH=$PATH:/<path_to_64-bit_SDK>/bin
```

- ☐ Issue the following command to set the `WLP_USR_DIR` to let the shell script know what server root to work from:

```
export WLP_USER_DIR=/<path_to_your_user_directory>
```

19 The auto-create will be attempted when you run the `ddlGen` utility. You can prevent this attempt by coding `createTables="false"` on the `<dataBaseStore>` element in the `server.xml`. Left uncoded, the default is `true` and it will try.

20 Unlike with Derby, the database does not need to be created ahead of time. However, each table will end up in a separate DB2-generated database, which may not be what your DBA prefers. If there's any doubt, then manually create using DDL as shown above.

21 The DDL-generation shell script you will run requires these to operate. localConnector is required for local JMX support.

For example, if your server is under `/u/user1/liberty/servers/server1`, then the `WLP_USER_DIR` value would be `/u/user1/liberty`.

- ☐ Now issue the following command:

```
/<Liberty_install_path>/bin/ddlGen generate <server name>
```

For example, if Liberty Profile is installed as `/usr/lpp/Liberty` and your server name is `server1`, the command would be:

```
/usr/lpp/Liberty/bin/ddlGen generate server1
```

- ☐ You should then see something like the following message:

```
CWWKD0107I: The requested DDL was generated in the
following directory: /u/user1/liberty/servers/server1/ddl
```

- ☐ There will be a file in that directory that contains the DB2 z/OS DDL for the creation of the tables[22]. The file is tagged ASCII, so it can be edited with OEDIT. Or you can download it to your workstation in binary for customization.
- ☐ Work with your DBA to create the tables as defined by the DDL[23].
- ☐ Remove the `<feature>localConnector-1.0</feature>` line from the `server.xml` and save the file[24].

Configure JSR 352 batch-1.0 support in server.xml

There are two pieces to this: (1) updating the `<featureManager>` section to include `batch-1.0` (this is easy: it's a one line update); and (2) providing the JDBC configuration elements so the JSR 352 support can communicate with the job repository data store. The syntax of the second depends on which data store format you chose.

Update <featureManager>

For the core JSR 352 support[25] one line is needed in the `<featureManager>` section.

Do the following:

- ☐ Edit the `server.xml` file
- ☐ Add the following line[26]:

```
<featureManager>
    <feature>servlet-3.1</feature>
    <feature>batch-1.0</feature>
</featureManager>
```

You should see the following message in the messages.log file:

```
CWWKF0012I: The server installed the following features:
                                                     [jdbc-4.1, jndi-1.0, batch-1.0].
```

JDBC definitions for JSR 352 job repository data store access

The syntax for this depends on the relational data store option you chose – in-memory, Derby, or a database product such as IBM DB2 z/OS. All three are shown next with notes to explain the configuration elements.

22 If a file is created but it is of zero length, then add `createTables="false"` to the `<databaseStore>` element in `server.xml`.
23 If you get SQLCODE -633, see "SQLCODE = -633, ERROR: THE DELETE RULE MUST BE CASCADE OR NO ACTION" on page 66.
24 This is optional … the feature could remain without affecting JSR 352 operations. We recommend removing it here just to keep the configuration consistent with the other examples that appear in this document where that feature is not shown.
25 For the REST and JBatch command line support the `batchManagement-1.0` element is needed. We cover that under “Configure batchManagement-1.0 support” on page 23.
26 This will already be there if you followed the steps to generate DDL using the `ddlGen` shell script.

In-Memory

Again, this is easy because there's nothing to do. The in-memory persistence model requires no JDBC definitions. Do the following:

- ☐ Go to "Validate batch-1.0 with supplied sample" on page 21.

Derby

Do the following:

- ☐ Stop the Liberty Profile server.
- ☐ Update your `server.xml` with the following XML[27]. See the notes that follow to be sure the XML matches the Derby database you created earlier.

```
 XML before this not shown
   :
<batchPersistence jobStoreRef="BatchDatabaseStore" />

<databaseStore id="BatchDatabaseStore"
  dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />

<library id="DerbyLib">
   <fileset dir="${server.config.dir}/resources/derby" />
</library>

<dataSource id="batchDB"
    type="javax.sql.XADataSource"
   jndiName="jdbc/batch">

 <jdbcDriver libraryRef="DerbyLib" />

 <properties.derby.embedded
   databaseName="${server.config.dir}/resources/BATCHDB"
   createDatabase="create"
   user="user"
   password="pass" />
</dataSource>
 :
XML after this not shown
```

Notes:

- The `fileset dir=` element points to the location where `derby.jar` and `derbytools.jar` files are located. This XML sample shows it pointing to `/resources/derby` under the server's configuration directory.
- The `databaseName=` element points to the location and directory name of the Derby database you created earlier. Again, this XML sample shows it pointing to the `/resources` directory with a database name of `BATCHDB`.

- ☐ Save the file.
- ☐ Restart the server.
- ☐ The tables will be created in the Derby database you created earlier.
- ☐ Go to "Validate batch-1.0 with supplied sample" on page 21.

DB2 z/OS

We will illustrate the use of JDBC Type 4 (network) access to DB2[28].

- ☐ Stop the Liberty Profile server.

27 See "Batch persistence – Derby" on page 68 for a full example of a `server.xml` configured for Derby batch persistence.
28 JDBC Type 2 (cross-memory) involves the Angel process and SAF SERVER profiles to enable access to authorized services. For simple validation JDBC Type 4 is easier. For production work you may wish to consider JDBC Type 2.

- ☐ Update your `server.xml` with the following XML[29]. See the notes that follow to understand how to customize the XML to your environment:

```
 XML before this not shown
  :
<batchPersistence jobStoreRef="BatchDatabaseStore" />

<databaseStore id="BatchDatabaseStore"
  dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />

<jdbcDriver id="DB2T4" libraryRef="DB2T4LibRef" />

<library id="DB2T4LibRef">
  <fileset dir="/<path>/jdbc/classes/"
  includes="db2jcc4.jar db2jcc_license_cisuz.jar sqlj4.zip" />
</library>

<authData id="batchAlias" user="<userid>" password="<password>" />

<dataSource id="batchDB"
    containerAuthDataRef="batchAlias"
    type="javax.sql.XADataSource"
    jdbcDriverRef="DB2T4">
  <properties.db2.jcc
   serverName="<host>"
   portNumber="<port>"
   databaseName="<location>"
   driverType="4" />
</dataSource>
 :
 XML after this not shown
```

Notes:

- The `fileset dir=` element points to the location where your DB2 z/OS JDBC drivers reside.
- Set `user=` and `password=` to a valid ID that has authority to access the DB2 system. The password can be encoded if you wish.
- Set `serverName=` to the host name where your DB2 DDF server is running.
- Set `portNumber=` to the port on which DDF is listening.
- Set `databaseName=` to the *location name* of your DB2 system.

- ☐ Save the file.
- ☐ Restart the server.
- ☐ Go to "Validate batch-1.0 with supplied sample" on page 21.

Validate batch-1.0 with supplied sample

At this point you should have your Liberty Profile server created, the job respository database created, and the `server.xml` updated. What's left is to validate this using the supplied SleepyBatchlet sample. Do the following:

- ☐ Go to following GitHub URL location:

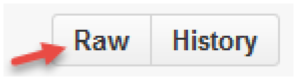
  `https://github.com/WASdev/sample.batch.sleepybatchlet`

- ☐ Click on the link for "SleepyBatchletSample-1.0.war":

SleepyBatchletSample-1.0.war

29 See "Batch persistence – DB2 z/OS" on page 69 for a full example of a `server.xml` configured for DB2 z/OS batch persistence.

- ☐ Click on the “Raw” button:



- ☐ It will prompt you to open or save the file. Save the file.
- ☐ Upload the file *in binary* to your z/OS system.
- ☐ Make sure the file permissions on the WAR file are at least `READ` for the ID under which the Liberty Profile server will run.
- ☐ Copy the file to the `/dropins` directory[30] of your Liberty Profile server configuration.
- ☐ Make sure the file permissions on the WAR file are at least `READ` for the ID under which the Liberty Profile server will run.
- ☐ Stop the server if it is running. (This is simply to have a new `messages.log` file to see things more clearly the first time.)
- ☐ Start the server.
- ☐ Look in the `messages.log` file. You should see something like this:

```
CWWKZ0018I: Starting application SleepyBatchletSample-1.0.
SRVE0169I: Loading Web Module: SleepyBatchletSample-1.0.
SRVE0250I: Web Module SleepyBatchletSample-1.0 has been bound to default_host.
CWWKT0016I: Web application available (default_host):
                                          http://<host>:9080/SleepyBatchletSample-1.0/
CWWKZ0001I: Application SleepyBatchletSample-1.0 started in 0.208 seconds.
CWWKF0008I: Feature update completed in 1.365 seconds.
CWWKF0011I: The server server1 is ready to run a smarter planet.
```

There will be several messages before this block of messages. The key we are looking for is the SleepyBatchletSample starting and the server itself indicating ready to run.

- ☐ Open a browser and send in the following URL:

`http://<host>:<port>/SleepyBatchletSample-1.0/sleepybatchlet?action=start`

Where `<host>` and `<port>` map to your system and your Liberty Profile server's configured port.

- ☐ After a moment[31], you should see something like the following:

```
Job started!

JobInstance: instanceId=0, jobName=sleepy-batchlet

JobExecution: executionId=0, jobName=sleepy-batchlet,
batchStatus=STARTED, createTime=2015-02-10 19:29:03.35,
startTime=2015-02-10 19:29:03.43, endTime=null,
lastUpdatedTime=2015-02-10 19:29:03.43, jobParameters={}
```

- ☐ By default that's going to run 15 seconds and the complete. Wait that period of time, then look in the `messages.log` file. You should see something like this:

```
SleepyBatchlet: process: entry
SleepyBatchlet: process: sleep for: 15
SleepyBatchlet: process: [0] sleeping for a second...
SleepyBatchlet: process: [1] sleeping for a second...
SleepyBatchlet: process: [2] sleeping for a second...
SleepyBatchlet: process: [3] sleeping for a second...
SleepyBatchlet: process: [4] sleeping for a second...
SleepyBatchlet: process: [5] sleeping for a second...
SleepyBatchlet: process: [6] sleeping for a second...
SleepyBatchlet: process: [7] sleeping for a second...
```

30 This is created when the Liberty Profile server is started the first time. If you don't see this directory, start the server.
31 If you are auto-creating tables in DB2 z/OS, this will take more than a moment. It may take up to a minute or more.

```
SleepyBatchlet: process: [8] sleeping for a second...
SleepyBatchlet: process: [9] sleeping for a second...
SleepyBatchlet: process: [10] sleeping for a second...
SleepyBatchlet: process: [11] sleeping for a second...
SleepyBatchlet: process: [12] sleeping for a second...
SleepyBatchlet: process: [13] sleeping for a second...
SleepyBatchlet: process: [14] sleeping for a second...
SleepyBatchlet: process: exit. exitStatus:
SleepyBatchlet:i=15;stopRequested=false
```

☐ Make sure it used the persistence store you intended[32]. Look for the following messages in `messages.log` based on your intended persistence store:

Memory:	`CWWKY0005I: The batch In-Memory persistence service is activated.`
	`CWWKY0008I: The batch feature is using persistence type In-Memory.`

Derby:	`CWWKY0005I: The batch JPA persistence service is activated.`
	`CWWKY0008I: The batch feature is using persistence type JPA.`
	` :`
	`DSRA8203I: Database product name : Apache Derby`
	`DSRA8204I: Database product version : 10.8.3.1 - (1452645)`
	`DSRA8205I: JDBC driver name   : Apache Derby Embedded JDBC Driver`
	`DSRA8206I: JDBC driver version   : 10.8.3.1 - (1452645)`

DB2:	`CWWKY0005I: The batch JPA persistence service is activated`
	`CWWKY0008I: The batch feature is using persistence type JPA`
	` :`
	`DSRA8203I: Database product name : DB2`
	`DSRA8204I: Database product version : DSN10015`
	`DSRA8205I: JDBC driver name   :`
	`IBM Data Server Driver for JDBC and SQLJ`
	`DSRA8206I: JDBC driver version   : 4.14.119`

If you intended to use Derby or DB2 and you saw the “In-Memory” messages, then there's something wrong with your `server.xml` and its pointer to the JSR 352 batch persistence store.

If you see this message:

```
SESN8501I: The session manager did not find a persistent storage
location; HttpSession objects will be stored in the local
application server's memory.
```

You can ignore that. That refers to HTTP session object storage, which is different from what the JSR 352 Java batch function uses.

Configure batchManagement-1.0 support

In this section we will discuss three topics related to the `batchManagement-1.0` feature:

Job logging This provides a mechanism for writing job logs to a separate location and file from the standard server output location.	Page 24
REST interface This is IBM-written function to submit, monitor and control the Java batch jobs that run in the JSR 352 runtime environment. This function implements a front-end on the JSR 352 JobOperator API.	Page 25
batchManager utility This is a Java utility that can be run from a command line environment to perform job management operations such as submit, monitor and control. This communicates with the REST interface to perform the functions.	Page 27

32 If it can't find a valid persistence store, it may assume in-memory and use it. So it may *look* like your JSR 352 implementation is working, but you may in fact not be using the Derby or DB2 store you intended. It's best to validate by checking messages in the log.

Go to the page indicated to read more.

Job logging

We start with this function of `batchManagement-1.0` because it's relatively simple to set up and use:

☐ Add the following line to the `<featureManager>` list of the `server.xml` for the Liberty Profile server:

```
<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batchManagement-1.0</feature>
  <feature>batch-1.0</feature>
</featureManager>
```

☐ Submit the SleepyBatchlet job as you did before.

☐ Look in the server's logs directory. You should now see:

```
/<server directory>/logs/
                          ├─/joblog
                          │  └── /sleepy-batchlet
                          │       └── /YYYY-MM-DD
                          │            └── /instance.#
                          │                 └── /execution.#
                          │                      part.#.log
                          └─/state
                          messages.log
```

Where `YYYY-MM-DD` is the date of the job execution, and the `#` symbols will be numbers based on the instance and execution IDs and log output part numbers.

☐ Look in the log file that was created (that will be in ASCII). You will see *something* like this:

```
[5/12/15 20:15:33:609 GMT] com.ibm.ws.batch.JobLogger
==========================================================
Started invoking execution for a job
 JobInstance id = 2
 JobExecution id = 2
 Job Name = sleepy-batchlet
 Job Parameters = {}
==========================================================

[5/12/15 20:15:33:609 GMT] com.ibm.ws.batch.JobLogger
==========================================================
For step name = step1
 New top-level step execution id = 2
==========================================================

[5/12/15 20:15:33:610 GMT] com.ibm.ws.batch.JobLogger
[5/12/15 20:15:48:622 GMT] com.ibm.ws.batch.JobLogger
[5/12/15 20:15:48:622 GMT] com.ibm.ws.batch.JobLogger
==========================================================
Completed invoking execution for a job
 JobInstance id = 2
 JobExecution id = 2
 Job Name = sleepy-batchlet
 Job Parameters = {}
 Job Batch Status = COMPLETED, Job Exit Status = COMPLETED
==========================================================
```

That is the default logging level.

☐ Submit the SleepyBatchlet job again. This will create a new job and execution instance. Now the file system structure will look like this:

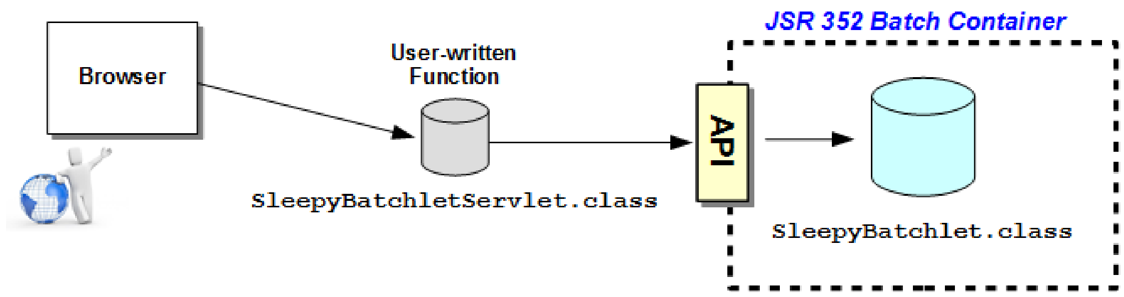
```
/<server directory>/logs/
                    ├─/joblog
                    │  └── /sleepy-batchlet
                    │       └── /YYYY-MM-DD
                    │            ├── /instance.#
                    │            │    └── /execution.#
                    │            │          part.#.log
                    │            └── /instance.#
                    │                 └── /execution.#
                    │                       part.#.log
                    └─/state
                    messages.log
```

That's the basics of job logging.

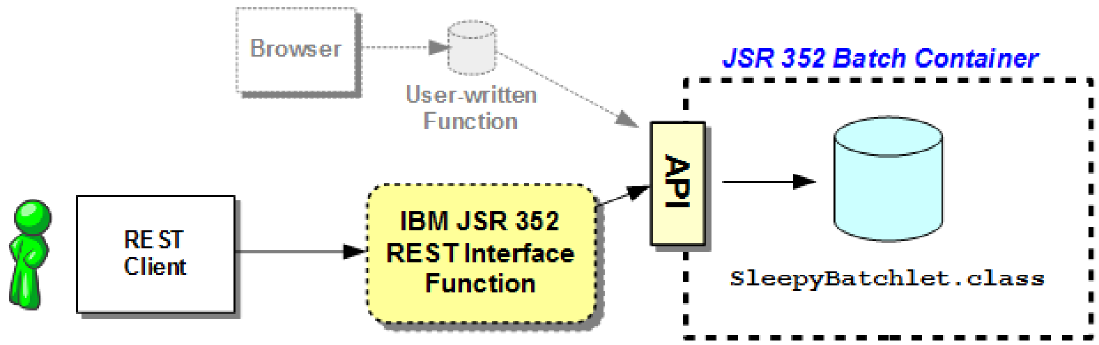
Note: Messages written by the application will be intercepted and sent to the job log only if `java.util.logging` is used, or another tool (such as Log4j) that uses `java.util.logging` as the backend. Messages written out using `println()` will *not* go to the job log, but *will* go the server STDOUT location.

REST Interface

In the earlier examples of running the SleepyBatchlet sample we used the servlet inside of the sample application to submit and control the jobs. That was an example of a user-written function that used the JSR 352 JobOperator API to control the jobs:



The IBM JSR 352 implementation comes with function that exposes a REST interface and uses the JSR 352 JobOperator API, so a user-written job control function is *not* required:



There are three security requirements imposed by this REST interface, all of which can be easily supplied during initial setup and validation:

- ***Encryption*** – the interface requires the client use SSL, which means an SSL handshake will be required, which means digital certificates will be involved. For initial setup and validation we will use a built-in function of Liberty Profile to keep this simple.
- ***Authentication*** – to use the REST interface, you will need to authenticate (commonly: a userid and password[33]). That implies having a userid and password in a “registry.” Here again, we will use a built-in function of Liberty Profile to keep this simple.
- ***Authorization*** – the userid used to authenticate is then checked to see if it has the authority to use the REST interface. This implies a “role,” which defines the authority granted to users. We can use a built-in function of Liberty Profile for this as well.

Note: in a production setting you will want to “harden” these security components by using either LDAP or SAF. Liberty Profile supports both. For the purposes of setup and validation we will keep things simple and code everything in the `server.xml`.

To establish the minimum security requirements to use the REST interface, do the following:

☐ Stop the server.

☐ Edit your server's `server.xml` file and add the following:

```
<server description="new server">

<featureManager>
  <feature>servlet-3.1</feature>
  <feature>batchManagement-1.0</feature>
  <feature>batch-1.0</feature>
  <feature>appSecurity-2.0</feature>    1
</featureManager>

<keyStore id="defaultKeyStore" password="Liberty"/>     2

<basicRegistry id="basic1" realm="jbatch">      3
   <user name="Fred" password="fredpwd" />
</basicRegistry>

<authorization-roles id="com.ibm.ws.batch">   4
   <security-role name="batchAdmin">
      <user name="Fred" />
   </security-role>
</authorization-roles>
  :
 (other server.xml elements from earlier follow here)
```

Notes:

1. This adds the application security feature.
2. This provides Liberty Profile the ability to establish an SSL session using certificates generated by Liberty Profile and stored in the server configuration directory structure.
3. This establishes a “registry” with one user – Fred – and Fred's password.
4. This provides the “batchAdmin” role[34] and grants Fred access to it.

33 You could also authenticate using a "client certificate." Here we will illustrate a userid and password to authenticate.
34 Three roles exist: `batchAdmin` (full authority); `batchSubmitter` (can submit and control their own jobs, but not others); and `batchMonitor` (can see status and logs of all jobs, but can't alter them in any way).

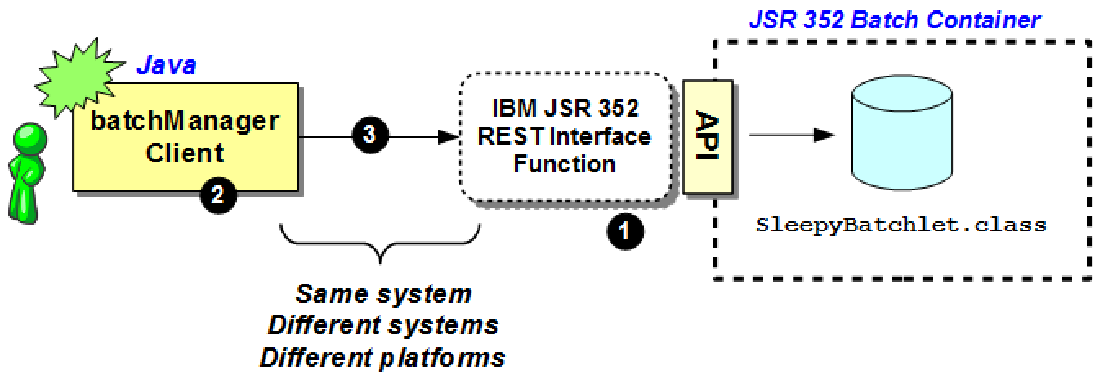
- ☐ Save the file.
- ☐ Start the server.

We expect most will use the supplied command line utility – `batchManager` – to drive the REST interface for job submission and control[35]. That is covered next.

However, if you're interested in seeing how a browser-based REST client would work, go to "Using a REST client to submit and monitor jobs" on page 76 for a review of using the Firefox REST client for that purpose.

batchManager utility

The batchManager utility makes use of the same REST interface illustrated in the previous section, and it provides a client with a command-line interface for submitting, monitoring and controlling batch jobs. The following picture illustrates this function:



Notes:

1. The same REST interface shown earlier. This means the `batchManagement-1.0` feature must be in place, as well as the security elements discussed on page 26.
2. The `batchManager` client is a Java client that provides a command line interface to the user, and turns the commands entered in the corresponding REST/JSON used by the IBM JSR 352 REST interface function.
3. The `batchManager` client sends REST/JSON over a network, and the client is platform-neutral Java. That means the client can run on the same system as the Liberty Profile server, a different system, or a completely different platform[36].

For *this* document our objective is to show how the `batchManager` client operates, so we will keep it simple. We will show use of `batchManager` on the same z/OS LPAR, and we'll use the Liberty-generated key file to allow SSL to work.

Important: What follows will work, but in a real-world environment you would *not* use the default Liberty generated key file, and you would *never* copy it like we're showing you here. This is simply an exercise to see batchManager work in the simplest way possible.

Do the following:

- ☐ Make sure the setup steps as illustrated under "REST Interface" starting on page 25 are complete. The `batchManager` client relies on the REST interface function being there.
- ☐ Open up a Telnet session to your z/OS system. Log in with whatever ID you wish.

35 The `batchManagerZos` utility also provides a command line interface, but it uses WOLA to interface with the IBM JSR 352 JobOperator. For more on `batchManagerZos`, go to page 39.

36 The same holds true for the client used for the REST interface function.

The `batchManager` client will attempt to establish an SSL connection to the Liberty Profile server, and to do that it needs access to the Liberty `key.jks` file[37]. That's located under the `/resources/security` directory in the server configuration:

```
/<user_directory>
           └─/servers
              └─/server1
                 └─/resources
                    └─/security
                          key.jks
```

We have two approaches open to us:

1 Simply point to the `key.jks` file in the configuration. That assumes the ID you use for `batchManager` has the UNIX permissions to directory search (`DIRSRCH`) down the path to the file, and has `READ` to the file itself.

2 Copy the `key.jks` file to a directory your logged-in ID has easy access to, and set the UNIX permissions so the ID has `READ`.

If the ID you will use to invoke `batchManager` has sufficient authority to do the first, then do that. Otherwise, follow the steps below to accomplish the second approach.

- ☐ Locate the `key.jks` file for the Liberty Profile z/OS server you are using. It is found under the `/resources/security` directory.
- ☐ Copy that file to a directory your logged-in ID has easy access to.
- ☐ Make sure the UNIX file permissions on that file allows at least `READ` for your logged-in ID.
- ☐ In the Telnet session, make sure the `JAVA_HOME` environment variable points to a valid Java SDK[38]:

  ```
  export JAVA_HOME=/<path_to_Java>
  ```
- ☐ In the Telnet session, export a JVM argument that points to the key.jks file you copied in the earlier step:

  ```
  export JVM_ARGS="-Djavax.net.ssl.trustStore=/<path>/key.jks"
  ```

  Where *`<path>`* points to the location where you copied the `key.jks` file, *or* *`<path>`* is the full path the `key.jks` file in the server configuration location if your ID has sufficient authority to `DIRSRCH` to it.
- ☐ Change directories[39] to where the batchManager utility is located:

  ```
  cd /<path>/bin
  ```

  Where *`<path>`* is where Liberty Profile is installed.
- ☐ Now, with `JAVA_HOME` and the `JVM_ARGS` set, you can invoke batchManager and submit the SleepyBatchlet job. Issue the following command (*all on one line*):

  ```
  ./batchManager submit --batchManager=localhost:9443 --user=Fred
       --password=fredpwd --applicationName=SleepyBatchletSample-1.0
                                  --jobXMLName=sleepy-batchlet.xml --wait
  ```

37 Instead, you may code --`trustSslCertificates` on the `batchManager` command. This eliminates this `key.jks` issue.
38 This can be set in the user's profile. What we're showing here is how to set the variable manually.
39 If you wish, you can put this location on the `PATH` and not change to the directory itself.

☐ You should see something like this:

```
[2015/05/14 09:41:35.152 -0400] CWWKY0101I: Job sleepy-batchlet
with instance ID 3 has been submitted.

[2015/05/14 09:41:35.154 -0400] CWWKY0106I: JobInstance:
{"jobName":"sleepy-
batchlet","instanceId":3,"appName":"SleepyBatchletSample-
1.0#SleepyBatchletSample-
1.0.war","submitter":"Fred","batchStatus":"STARTING","jobXMLName"
:"sleepy-batchlet.xml","instanceState":"SUBMITTED"}
```

Which indicates the job is submitted and running.

The `-wait` parameter means the client will wait until the job finishes, report that to you, and then end:

```
[2015/05/14 09:42:05.252 -0400] CWWKY0105I: Job sleepy-batchlet
with instance ID 3 has finished. Batch status: COMPLETED. Exit
status: COMPLETED

[2015/05/14 09:42:05.253 -0400] CWWKY0107I: JobExecution:
{"jobName":"sleepy-batchlet","executionId":3,"instanceId":3,
"batchStatus":"COMPLETED","exitStatus":"COMPLETED","createTime":"
2015/05/14 13:41:35.094 +0000","endTime":"2015/05/14 13:41:50.214
+0000","lastUpdatedTime":"2015/05/14 13:41:50.214
+0000","startTime":"2015/05/14 13:41:35.117
+0000","jobParameters":
{},"restUrl":"https://myhost.com:9443/ibm/api/batch","serverId":"
localhost//u/libserv/liberty/server1","logpath":"/u/libserv/liber
ty/servers/server1/logs/joblogs/sleepy-batchlet/2015-05-
14/instance.3/execution.3/","stepExecutions":
[{"stepExecutionId":3,"stepName":"step1","batchStatus":"COMPLETED
","exitStatus":"SleepyBatchlet:i=15;stopRequested=false","stepExe
cution":"https://localhost:9443/ibm/api/batch/jobexecutions/3/ste
pexecutions/step1"}]}
```

☐ Explore the “help” facility of batchManager … issue the following command:

```
./batchManager --help
```

You should see:

```
Usage: batchManager {help|submit|stop|restart|status|getJobLog|
listJobs|purge} [options]
```

```
Actions:

    help
        Print help information for the specified action.

    submit
        Submit a new batch job.

    stop
        Stop a batch job.

    restart
        Restart a batch job.

    status
        View the status of a job.

    getJobLog
        Download the joblog for a batch job.
    listJobs
        List job instances.

    purge
        Purge all records and logs for a job instance.

Options:
        Use help [action] for detailed option information of each
        action.
```

☐ Invoke "help" on the submit action:

```
./batchManager --help submit
```

You will see:

```
Usage:
        batchManager submit [options]

Description:
        Submit a new batch job.

Required:
    --user=[username]
        The username for logging in to the batch manager.

    --password[=pwd]
        The password for logging in to the batch manager. If no value is
        defined you will be prompted.

    --batchManager=[host]:[port],[host2]:[port2],...
        The host and port of the batch manager REST API. You can specify
        multiple targets for high availability and fail-over. Targets are
        delimited by a comma ','.

    --jobXMLName=[jobXMLName]
        The name of the job XML describing the job.

      :
   (other output removed to save space in this document)
      :
```

You can do the same for the other actions[40]. This provides you a way to get the syntax of the commands.

Using the BonusPayout chunk application sample

The SleepyBatchlet application was an example of a *batchlet* step[41]. It was easy to start with for validation because it required no input or output information, and could be submitted without any job parameters.

The other JSR 352 batch step type is a *chunk* step, which is looping, transactional process most people think of when they think of "batch." The BonusPayout sample provides this.

Unlike SleepyBatchlet, the BonusPayout sample has some input and output requirements:

- A UNIX file, which the application will auto-create and populate with data
- A relational database table (Derby or DB2), which you *must* create ahead of time.

Create the ACCOUNT database table

The database table this batch application uses is relatively simple. It's a single table with four columns. This table may reside in Derby or DB2 z/OS. Both are illustrated below.

40 The other actions were listed out with the simple `batchManager --help` command.
41 See "JSR 352 step types" on page 9 for an overview of the JSR 352 job step types.

If Derby

Do the following:

- ☐ Review the process used for the creation of the JobRepository database with Derby back on page 16. It'll be the same process, just a different DDL file pointed to.
- ☐ The DDL for this table is shown here:

```
CONNECT 'jdbc:derby:{database};create=true';

DROP TABLE "{database}"."ACCOUNT";

CREATE TABLE "{database}"."ACCOUNT"   (
           "ACCTNUM" INTEGER NOT NULL,
           "BALANCE" INTEGER NOT NULL,
           "INSTANCEID" BIGINT NOT NULL,
           "ACCTCODE" VARCHAR(30) )
          ;

ALTER TABLE "{database}"."ACCOUNT"
   ADD CONSTRAINT "ACCOUNT_PK" PRIMARY KEY
         ("ACCTNUM", "INSTANCEID");

COMMIT WORK;

DISCONNECT;
```

- ☐ Create a text file on your workstation. Copy/paste the DDL into the text file and change the four instances of `{database}` to some value of your choosing … such as `BONUSDB`.
- ☐ Save the file and upload *in binary* to your server.
- ☐ Use the same process as illustrated started on page 16 to create a separate Derby database for this table.
- ☐ Be sure to recursively `chown` the database directory structure so it's owned by the ID that Liberty Profile runs under.

If DB2 z/OS

Do the following:

- ☐ Work with your database administrator to create a database and table using the following DDL:

```
SET CURRENT SQLID = '{sqlid}';
SET CURRENT SCHEMA = '{schema}';

-- DROP TABLE ACCOUNT;
-- DROP DATABASE {database};
-- COMMIT;

CREATE DATABASE {database}
  BUFFERPOOL {bufferpool}
  INDEXBP {bufferindex}
  STOGROUP {storegroup}
  CCSID UNICODE;

CREATE TABLESPACE {tablespace} IN {database}
  USING STOGROUP {storegroup}
  PRIQTY 1500
  SECQTY -1
  ERASE NO
  DEFINE NO
  SEGSIZE 32
  LOCKSIZE ROW;

GRANT USE OF TABLESPACE {database}.{tablespace} TO PUBLIC;

CREATE TABLE ACCOUNT(
  ACCTNUM INTEGER NOT NULL,
  BALANCE INTEGER NOT NULL,
  INSTANCEID BIGINT NOT NULL,
  ACCTCODE VARCHAR(30),
  CONSTRAINT ACCOUNT_PK PRIMARY KEY (ACCTNUM,INSTANCEID)
  ) IN {database}.{tablespace};

CREATE UNIQUE INDEX ACCT_IDX ON ACCOUNT(ACCTNUM,INSTANCEID)
  USING STOGROUP {storegroup}
  PRIQTY 720
  SECQTY -1
  BUFFERPOOL {bufferpool};

COMMIT;
```

Update the placeholders with your values and capture them here:

Variable	*Your value*
`{sqlid}`	
`{schema}`	
`{database}`	
`{storegroup}`	
`{tablespace}`	
`{bufferpool}`	
`{bufferindex}`	

Update the server.xml for application access to the "account" table

This involves adding additional XML to provide the JDBC access for the Bonus application. But the XML you add to the file depends on what's already there for the JSR 352 persistence[42].

To assist with determining what to add to the `server.xml` file, a set of samples is provided in the back of the document. Do the following:

- ☐ Use this table for a pointer to the sample that applies to your setup:

	JSR 352 Persistence	
	Derby	*DB2 z/OS*
Bonus: *Derby*	Page 70	Page 73
Bonus: *DB2 z/OS*	Page 71	Page 74

- ☐ Review the sample for your environment. Those samples all have the JSR 352 persistence definitions first, then the BonusPayout sample second.
- ☐ Copy the relevant XML from the sample and paste into your `server.xml`.

42 If JSR 352 is using in-memory, then there's nothing in the `server.xml` defining JDBC access. The updates for Bonus application are net new. Configure the JDBC definitions for Derby or DB2 from the examples in the back.

- ☐ Modify as needed, paying particular attention to the JNDI name, the database name (when Derby) and the specific DB2 z/OS definitions (when DB2).
- ☐ Save the file.

For example, if you used Derby for the JSR 352 persistence and DB2 z/OS for Bonus, then the sample would be on page 71, and the update would be as follows. The JSR 352 persistence definitions highlighted in gray; the Bonus definitions in this color.

XML before this not shown

```
 :
<batchPersistence jobStoreRef="BatchDatabaseStore" />

<databaseStore id="BatchDatabaseStore"
  dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />

<library id="DerbyLib">
   <fileset dir="${server.config.dir}/resources/derby" />
</library>

<dataSource id="batchDB"
   jndiName="jdbc/batch"    ← JNDI for Java Batch
   type="javax.sql.XADataSource">

 <jdbcDriver libraryRef="DerbyLib" />

 <properties.derby.embedded
   databaseName="${server.config.dir}/resources/BATCHDB"
   createDatabase="create"
   user="user"
   password="pass" />
</dataSource>

<jdbcDriver id="DB2T4" libraryRef="DB2T4LibRef" />

<library id="DB2T4LibRef">
  <fileset dir="/<path>/jdbc/classes/"
  includes="db2jcc4.jar db2jcc_license_cisuz.jar sqlj4.zip" />
</library>

<dataSource id="bonusDB"
    jndiName="jdbc/bonus"     ← JNDI for BonusPayout sample
    type="javax.sql.XADataSource"
    jdbcDriverRef="DB2T4">
  <properties.db2.jcc
    serverName="<host>"
    portNumber="<port>"
    databaseName="<location_name>"
    user="<userid>"
    password="<password>"
    driverType="4" />
</dataSource>
:
```

XML after this not shown

Get the BonusPayout sample from Github

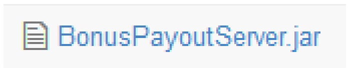
This application is bundled as part of a bigger file that includes Liberty Profile itself. It takes a bit more to get to the WAR file, but not much more. Do the following:

- ☐ Go to the following GitHub URL:

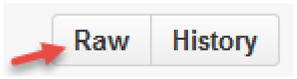
https://github.com/WASdev/sample.batch.bonuspayout

- ☐ Click on the "PrePackaged" link:

- ☐ Click on the "BonusPayoutServer.jar" link:



- ☐ Click on the "Raw" button:



- ☐ You will be prompted to open or save the file. Save the file. It is 40MB in size.
- ☐ Use a zip tool to open the JAR file. In that JAR you will see a file named `BonusPayout-1.0.war`. Extract that file from the JAR.

Deploy the batch application and invoke

With the database created (either Derby or DB2) and the `server.xml` updated to included a valid `<dataSource>` pointing to where the BonusPayout table resides, now you're ready to deploy the application and invoke it.

Do the following:

- ☐ Upload the `BonusPayout-1.0.war` file to the z/OS system. Set the file permissions so the ID under which the Liberty Profile server runs has `READ` to it.
- ☐ Copy the `BonusPayout-1.0.war` file into the `/dropins` directory.
- ☐ You should see something like this:

```
CWWKZ0001I: Application BonusPayout-1.0 started in 1.184 seconds.
```

- ☐ To submit this we'll use the `batchManager` utility as described earlier under "batchManager utility" starting on page 27. Submitting BonusPayout requires a slightly different set of parameters from SleepyBatchlet[43], but the general approach is the same.
- ☐ Make sure the setup activities for invoking batchManager are in place, as described earlier. Go back and review the directions starting on page 27:
  - ○ Telnet session open and the ID you logged in with has access to the Liberty Profile `key.jks` file … either because it has authority to directory search into the Liberty configuration path, or because you copied the `key.jks` to another location.
  - ○ Export `JAVA_HOME` with a valid pointer to the 64-bit SDK
  - ○ Export `JVM_ARGS` with the pointer to the `key.jks` file
  - ○ Change directories to the Liberty install `/bin` directory
- ☐ From your `server.xml` file, capture the JNDI name used for access to the account table … such as `jdbc/bonus`, or whatever you coded that as.

Account Table JNDI:	

43 For BonusPayout we need to pass in the JNDI name used for the relational table access, and we need to pass in the table name.

- ☐ Capture here the table name you created for the BonusPayout application:

Derby `database:tablename`	
DB2 `schema:tablename`	

- ☐ Compose the following in Notepad *as one line*, then paste into your Telnet session and submit:

```
./batchManager submit --batchManager=localhost:9443
                                    --user=Fred --password=fredpwd
                                 --applicationName=BonusPayout-1.0
                             --jobXMLName=SimpleBonusPayoutJob.xml
                                  --jobParameter=dsJNDI=jdbc/bonus
                  --jobParameter=tableName=BONUSDB.ACCOUNT --wait
```

Where the two --`jobParameter=` values are customized to your environment.

- ☐ You should see a job submission message like this:

```
[2015/06/05 19:00:11.532 -0400] CWWKY0101I: Job
SimpleBonusPayoutJob with instance ID 2 has been submitted.
[2015/06/05 19:00:11.535 -0400] CWWKY0106I: JobInstance:
{"jobName":"SimpleBonusPayoutJob","instanceId":2,"appName":"Bonus
Payout-1.0#BonusPayout-1.0.war","submitter":"Fred",
"batchStatus":"STARTING","jobXMLName":"SimpleBonusPayoutJob.xml",
"instanceState":"SUBMITTED"}
```

- ☐ The --`wait` parameter means `batchManager` will wait until the job completes. When it completes you will see a lengthy message indicating `COMPLETED`.

- ☐ Look in the server root directory (the same directory where `server.xml` resides). You should see a file there with an extension of "csv." That's the output file from the first step of the job[44]. That file is in ASCII. The contents of that file will look something like this:

```
0,441,CHK
1,401,CHK
2,125,CHK
3,823,CHK
4,728,CHK
  :
998,407,CHK
999,457,CHK
```

- ☐ If you look in the database table you'll find something like this:

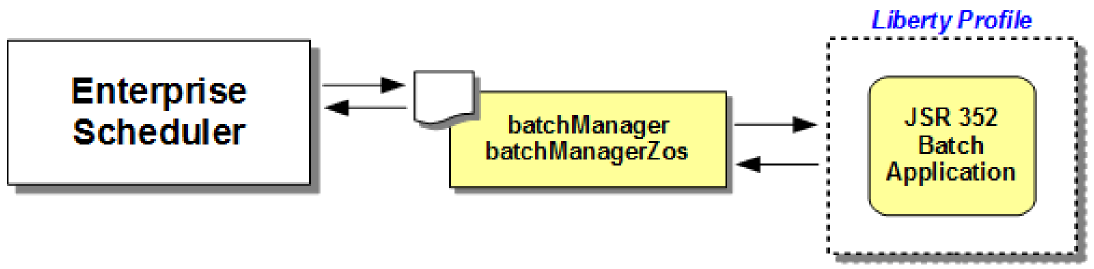
```
---------+---------+---------+---------+---------+---------
    ACCTNUM      BALANCE                INSTANCEID  ACCTCODE
---------+---------+---------+---------+---------+---------
          0          541                         2  CHK
          1          501                         2  CHK
          2          225                         2  CHK
          3          923                         2  CHK
          4          828                         2  CHK
          :           :                          :   :
```

44 The jobParameter `generateFileNameRoot` provides a way to specify a path and prefix for this file.

The default “bonus” amount is 100, which you can see by comparing the balance in the CSV file with the balance in the database table. The second step of the job took the starting balance, added the default bonus amount (100) and wrote it to the database.

External scheduler support

Eventually some form of scheduler will be used to initiate the jobs and track the completion of those jobs. For the IBM JSR 352 function running inside Liberty Profile, two functions are provided:



batchManager – this is a command line interface that uses the REST interface to submit and monitor jobs.

You used the `batchManager` function earlier (“batchManager utility” on page 27). There you *manually* invoked batchManager and submitted a job. To use with a scheduler you could write a shell script to issue the command, or write a JCL job that uses BPXBATCH to create a shell and issue the command.

batchManagerZos – this is a native z/OS function that uses WOLA (WebSphere Optimized Local Adapters) to communicate with the Liberty Profile z/OS server. This function can be invoked at a command prompt, with a shell script[45], or from JCL using BPXBATCH.

The following table summarizes the two functions:

`batchManager`	• Instatiates a JVM with each invocation • Uses REST and TCP/IP to communicate with the Liberty Profile • May be used from the same LPAR or a different LPAR, or from a different server or server platform • Invoke manually, from a shell script, or from JCL using BPXBATCH
`batchManagerZos`	• Native z/OS function used with Liberty Profile z/OS only • Does **not** instantiate a JVM with usage • Uses cross-memory WOLA to communicate with Liberty Profile z/OS • Must be used on the same LPAR as the JSR 352 Liberty server • Invoke manually, from a shell script, or from JCL using BPXBATCH

In this section we will explore `batchManagerZos`.

Determine readiness to run WOLA

Using WOLA requires several things to be in place, including the angel process and a few SAF profiles. Let's check to see the status of your environment. Do the following:

☐ Look at the top of your messages.log file. Messages there indicate some key things about readiness to use WOLA:

Not Ready:

```
CWWKE0001I: The server server1 has been launched.
CWWKB0102I: This server is not authorized to connect to the  1
```

45 The execution of batchManagerZos must be on the same LPAR as the Liberty Profile z/OS server running IBM JSR 352. That is because it uses WOLA, and WOLA is a cross-memory (same LPAR) technology.

```
              angel process.  No authorized services will be loaded.
CWWKB0104I: Authorized service group LOCALCOM is not available. 2
CWWKB0104I: Authorized service group WOLA is not available.
CWWKB0104I: Authorized service group PRODMGR is not available.
CWWKB0104I: Authorized service group SAFCRED is not available.
CWWKB0104I: Authorized service group TXRRS is not available.
CWWKB0104I: Authorized service group ZOSDUMP is not available.
CWWKB0104I: Authorized service group ZOSWLM is not available.
```

or

```
CWWKE0001I: The server server1 has been launched.
CWWKB0101I: The angel process is not available.  3
                            No authorized services will be loaded.
CWWKB0104I: Authorized service group LOCALCOM is not available.
CWWKB0104I: Authorized service group WOLA is not available.
CWWKB0104I: Authorized service group PRODMGR is not available.
CWWKB0104I: Authorized service group SAFCRED is not available.
CWWKB0104I: Authorized service group TXRRS is not available.
CWWKB0104I: Authorized service group ZOSDUMP is not available.
CWWKB0104I: Authorized service group ZOSWLM is not available.
```

Ready:

```
CWWKE0001I: The server server1 has been launched.
CWWKB0103I: Authorized service group LOCALCOM is available.  4
CWWKB0103I: Authorized service group WOLA is available.
CWWKB0104I: Authorized service group PRODMGR is not available.
CWWKB0104I: Authorized service group SAFCRED is not available.
CWWKB0104I: Authorized service group TXRRS is not available.
CWWKB0104I: Authorized service group ZOSDUMP is not available.
CWWKB0104I: Authorized service group ZOSWLM is not available.
```

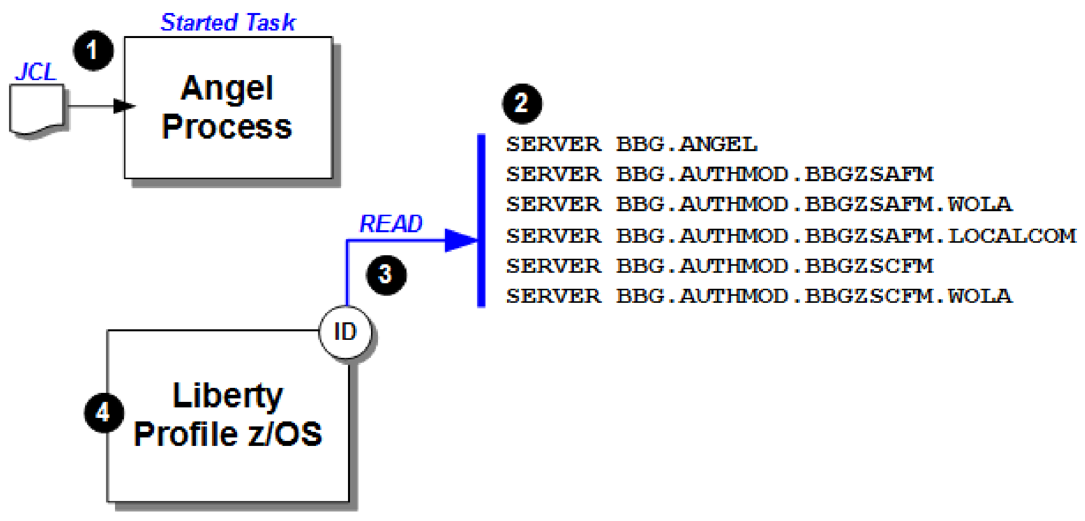
Notes:

1. This message indicates the angel process is up, but the Liberty Profile server ID does not have READ to the required SAF profiles.
2. The two required authorized service groups (`LOCALCOM` and `WOLA`) are showing *not* available because the server lacks READ to the SAF profiles.
3. The angel process is not started. WOLA can not be used without the angel.
4. The angel process is started *and* the server has READ to the key SAF profiles to enable access to the `LOCALCOM` and `WOLA` authorized service groups.

☐ If your environment is ready to use WOLA, then go to "Use batchManagerZos" on page 39. Otherwise, continue to the next section ("Enable WOLA support").

Enable WOLA support

The objective of this section is to achieve the following (see the notes that follow):


Started Task
JCL
1
Angel Process
2
SERVER BBG.ANGEL
SERVER BBG.AUTHMOD.BBGZSAFM
SERVER BBG.AUTHMOD.BBGZSAFM.WOLA
SERVER BBG.AUTHMOD.BBGZSAFM.LOCALCOM
SERVER BBG.AUTHMOD.BBGZSCFM
SERVER BBG.AUTHMOD.BBGZSCFM.WOLA
READ
3
ID
4
Liberty Profile z/OS

Notes:

1. An angel process must be started on the z/OS LPAR.

   Note: You may already have an angel process present. If z/OS 2.1 and z/OSMF in use, then an angel is present[46]. You may use that angel. Only one angle per LPAR is required.

2. The six SAF `SERVER` profiles shown are required to be in place.
3. The ID under which the Liberty Profile z/OS server runs must have `READ` to the six SAF `SERVER` profiles.
4. The Liberty Profile z/OS server started with the `messages.log` file showing the key indicators of WOLA availability.

Do the following:

- ☐ Determine if an angel process is present
  - ○ If yes, verify the level of Liberty Profile z/OS the angel uses is at least 8.5.5.2. If not, then work with your system programmer to update the running angel to 8.5.5.2 or higher.
  - ○ If no, then see the WP102110 Techdoc at `ibm.com/support/techdocs` for instructions on creating and starting the angel process.
- ☐ Once the angel process is in place and started, work with your security administrator to create the following six SAF profiles and grant your Liberty Profile z/OS server ID has `READ`. The commands for RACF are shown here:

```
RDEFINE SERVER BBG.ANGEL UACC(NONE) OWNER(SYS1)
PERMIT  BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(<server_ID>)

RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE) OWNER(SYS1)
PERMIT  BBG.AUTHMOD.BBGZSAFM -
        CLASS(SERVER) ACCESS(READ) ID(<server_ID>)

RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.WOLA UACC(NONE) OWNER(SYS1)
PERMIT  BBG.AUTHMOD.BBGZSAFM.WOLA -
        CLASS(SERVER) ACCESS(READ) ID(<server_ID>)

RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.LOCALCOM UACC(NONE) OWNER(SYS1)
```

46 Starting with z/OS 2.1 the z/OSMF function began using Liberty Profile as its server runtime. Since z/OSMF makes extensive use of authorized services, an angel process was required. The angel is place for z/OSMF may be used by any other Liberty Profile z/OS servers on the LPAR. Check to make sure the angel is at least at the 8.5.5.2 level of Liberty Profile. If not, then work with your z/OS system programmer to update z/OSMF to Liberty Profile z/OS 8.5.5.2 or higher. That level of angel is required to use WOLA.

```
PERMIT   BBG.AUTHMOD.BBGZSAFM.LOCALCOM -
         CLASS(SERVER) ACCESS(READ) ID(<server_ID>)

RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM UACC(NONE) OWNER(SYS1)
PERMIT   BBG.AUTHMOD.BBGZSCFM -
         CLASS(SERVER) ACCESS(READ) ID(<server_ID>)

RDEFINE SERVER BBG.AUTHMOD.BBGZSCFM.WOLA UACC(NONE) OWNER(SYS1)
PERMIT   BBG.AUTHMOD.BBGZSCFM.WOLA -
         CLASS(SERVER) ACCESS(READ) ID(<server_ID>)

SETROPTS RACLIST(SERVER) REFRESH
```

- ☐ Restart the Liberty Profile z/OS server and look for the key messages:

```
CWWKE0001I: The server server1 has been launched.
CWWKB0103I: Authorized service group LOCALCOM is available.
CWWKB0103I: Authorized service group WOLA is available.
CWWKB0104I: Authorized service group PRODMGR is not available.
  :
```

Use batchManagerZos

With the WOLA support made ready, you may take the next steps to using the batchManagerZos function. Do the following:

- ☐ Update the `server.xml` file with the following:

```
<!-- Enable features -->
<featureManager>
    <feature>servlet-3.1</feature>
    <feature>batch-1.0</feature>
    <feature>batchManagement-1.0</feature>
    <feature>appSecurity-2.0</feature>
    <feature>zosLocalAdapters-1.0</feature>
</featureManager>

<zosLocalAdapters wolaGroup="LIBERTY"
                  wolaName2="BATCH"
                  wolaName3="MANAGER"/>

<keyStore id="defaultKeyStore" password="Liberty"/>
  :
```

The `zosLocalAdapters-1.0` feature enables the use of WOLA for this server.

The `<zosLocalAdapters>` XML defines the WOLA three-part name for this server. You will use this three-part name with the `batchManagerZos` command to indicate which server WOLA should link to and communicate.

- ☐ Save the file.
- ☐ Those changes will trigger a series of dynamic updates to your Liberty Profile server. Look in the `messages.log` file for the following message:

```
CWWKB0501I: The WebSphere Optimized Local Adapter channel
registered with the Liberty profile server using the following
name: LIBERTY BATCH MANAGER
```

That indicates the server is using WOLA and is "listening" on that three-part name. Presence of that message indicates all the other things are in place – the angel process, and the SERVER profiles with READ access to this server.

- ☐ One more SAF profile is needed – this one CBIND – which will permit the ID[47] you use to run batchManagerZos the ability to use WOLA and communicate with the server:

```
RDEFINE CBIND BBG.WOLA.LIBERTY.BATCH.MANAGER -
        UACC(NONE) OWNER(SYS1)
PERMIT BBG.WOLA.LIBERTY.BATCH.MANAGER -
        CLASS(CBIND) ACCESS(READ) ID(<ID>)

SETROPTS RACLIST(CBIND) REFRESH
```

- ☐ Open a Telnet session with your system and log in with the ID you granted READ to the CBIND profile.
- ☐ Change directories to:

  `/<Liberty_install_mount>/lib/native/zos/s390x`

  If you list the contents of that directory, you should see batchManagerZos:

```
batchManagerZos      bbgzcsl      bbgzsufm
bbgzadrm             bbgzsafm     bboacall
bbgzafsm             bbgzscfm     libzNativeServices.so
bbgzangl             bbgzsrv      nls
```

- ☐ To test basic WOLA connectivity, use the ping command:

```
./batchManagerZos ping --batchManager=LIBERTY+BATCH+MANAGER
```

  If *successful*, it will simply return the command prompt. No other positive indicator is offered.

  If it *fails*, you will get a lengthy error message. Work through the following actions to determine the problem:

  - ○ Make sure the angel is active and your server has the proper access to the angel and authorized services. Look for the messages in messages.log shown earlier to indicate success.
  - ○ Make sure the WOLA three-part name specified in the server.xml file is in effect. Look for the message in the messages.log file to indicate the three-part name is in use for this server.
  - ○ Make sure the WOLA three-part name used on the ping command exactly matched the three-part name used by the server.
  - ○ Make sure the CBIND SAF profile is in effect, and the ID used to issue the batchManagerZos command has READ access to the CBIND profile

- ☐ Update the server.xml file with the following:

```
<authorization-roles id="com.ibm.ws.batch">
   <security-role name="batchAdmin">
      <special-subject type="EVERYONE"/>
      <user name="Fred" />
      <user name="USER1" />
   </security-role>
</authorization-roles>
```

That will grant the ID you used to invoke batchManagerZos access to the batch administrator role[48]. Without that update, your request will be reject because the ID

47 For example, if you Telnet into your system to run batchManagerZos, then the ID you log into Telnet with. Or if you run batchManagerZos from a JCL job, then the ID under which that job operates.

does not have authority to submit the job.

- ☐ Save the file. Liberty will dynamically update with the change.
- ☐ Assuming you've deployed the SleepyBatchlet application from earlier, you're ready to submit a job using `batchManagerZos`. Enter the following command as *one line*:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
                        --applicationName=SleepyBatchletSample-1.0
                            --jobXMLName=sleepy-batchlet.xml --wait
```

You should something like this:

```
INFO: CWWKY0101I: Job sleepy-batchlet with instance ID 9 has been
submitted.
INFO: CWWKY0106I: JobInstance:{"jobName":"sleepy-batchlet",
"instanceId":9,"appName":"SleepyBatchletSample-1.0
#SleepyBatchletSample-1.0.war","submitter":"null","batchStatus":
"STARTING","jobXMLName":"sleepy-batchlet.xml",
"instanceState":"SUBMITTED"}
```

Because the --`wait` was on the command, `batchManagerZos` will wait for the job to complete. When it does you will then see something like this:

```
INFO: CWWKY0105I: Job sleepy-batchlet with instance ID 9 has finished.
Batch status: COMPLETED. Exit status: COMPLETED

INFO: CWWKY0107I: JobExecution:{"jobName":"sleepy-
batchlet","executionId":9,"instanceId":9,"batchStatus":"COMPLETED","exitS
tatus":"COMPLETED",
  :
[{"stepExecutionId":11,"stepName":"step1","batchStatus":"COMPLETED","exit
Status":"SleepyBatchlet:i=15;stopRequested=false"}]}
```

- ☐ Issue the following command:

```
./batchManagerZos help
```

You should see:

```
Usage: batchManagerZos {help|ping|submit|stop|restart|status}
[options]

Actions:

   help
       Use help [action] for detailed option information.

   ping
       'Ping' the batch manager to test connectivity.

   submit
       Submit a new batch job.

   stop
       Stop a batch job.

   restart
       Restart a batch job.

   status
       View the status of a job.
```

48 There is a way to have SAF protect access using EJBROLE profiles. Search the Knowledge Center for keyword string `twlp_batchManagerZos` for more on the EJBROLE profiles the ID would need access to.

```
Options:
        Use help [action] for detailed option information.
```

- ☐ Now issue the command:

  `./batchManagerZos help submit`

  You should get a detailed listing of the parameters on the submit command. You may use the `help [action]` command to see details on the other options.

- ☐ You can execute `batchManagerZos` using JCL and `BPXBATCH` with a job that looks like this:

```
//ZBATCH JOB (ACCTNO,ROOM),REGION=0M,
// USER=<user>,PASSWORD=<password>
//SUBMIT   EXEC PGM=BPXBATCH
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//STDPARM  DD  *
PGM
/<install_path>/lib/native/zos/s390x/batchManagerZos
submit
--batchManager=LIBERTY+BATCH+MANAGER
--applicationName=SleepyBatchletSample-1.0
--jobXMLName=sleepy-batchlet.xml
--wait
/*
```

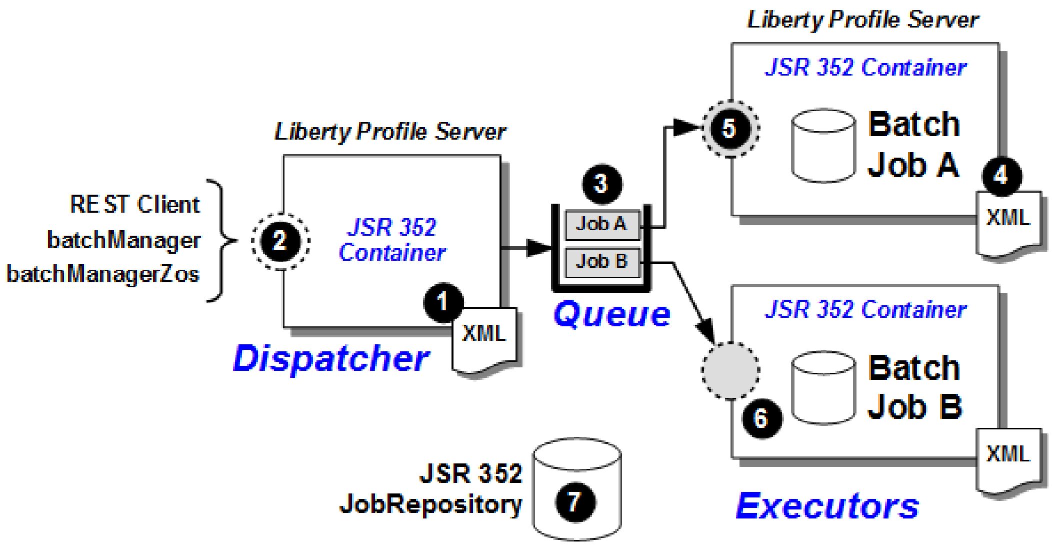
  However, some notes about this:

  - There is a known issue with the return code. The return code `BPXBATCH` will issue for a successful completion is `768`. This is due to an offset issue[49]. The submission works, it's just the return code is `RC=768` rather than `RC=0`.

  - For those who are wondering if the `batchManagerZos` module can be copied out to a PDSE and run with `PGM=`, the answer is "not yet." If attempted, that will result in an `OC4` abend.

Multi-JVM support

This function is best illustrated with a picture (see notes that follow):



49 See this link for more on the offset issue.

Notes:

1. A Liberty Profile server is configured as a "Dispatcher." This means it will accept job submissions through the interfaces we saw earlier, but the jobs will not run there. Configuring a server as a Dispatcher is done in the `server.xml`.
2. The Dispatcher will accept job submissions in the same way we illustrated earlier – through the REST interface using a client or the `batchManager` command line client, or on z/OS using the `batchManagerZos` command line client.
3. The Dispatcher will put a job submission message on a queue. The queue may be in a product such as IBM MQ, or it may be a queue in the messaging function of Liberty Profile itself[50].
4. A Liberty Profile server is configured as an "Executor." This means it will execute jobs submitted through a Dispatcher. Configuring a server as an Executor is done in the `server.xml`.
5. The Executor is configured to listen on the queue on which the Dispatcher puts the job submission message. In addition, the Executor can be configured with criteria to use when selecting messages off the queue. For example, in this picture the Executor can be configured to look for job submission messages for Job A only.

   Note: this is done with a "message selector," which is an XML element that identifies criteria the Executor will use when it looks at the messages in the queue. It picks up only those messages that match the configured message selector criteria.

   A simple example is this:

   ```
   messageSelector="com_ibm_ws_batch_applicationName = 'SleepyBatchletSample-1.0'"
   ```

   That tells the server to pick up job submission messages for the SleepyBatchlet application, but no others.
6. Other servers may be configured as Executors as well, each listening on the queue and looking for job submission messages it is configured to look for. For example, in this picture the second Executor server can be configured to look for job submission messages for Job B only.
7. The Dispatcher and Executor servers require common knowledge of the jobs submitted, and that's done through a *shared* JobRepository. That means the in-memory option can't be used (in-memory is accessible by only one server), and Derby is more involved to used in sharing mode[51]. But a database product such as IBM DB2 *can* be used.

There are several benefits of this design:

- Integration with an enterprise scheduler is easier since the invocation of the `batchManager` or `batchManagerZos` utilities only need to communicate with one server, not each server where the job is located. This gives you greater flexibility to move batch applications among your defined Executor servers.
- Jobs can be submitted to the Dispatcher even though the Executor where the batch application is deployed is not up. The job submission message will sit in the queue. When the Executor server comes active, it will read the queue, select its messages and begin job execution.
- Jobs can be deployed into servers based on importance of the job, with the Executor servers access to system resources managed by z/OS WLM based on service class goals.

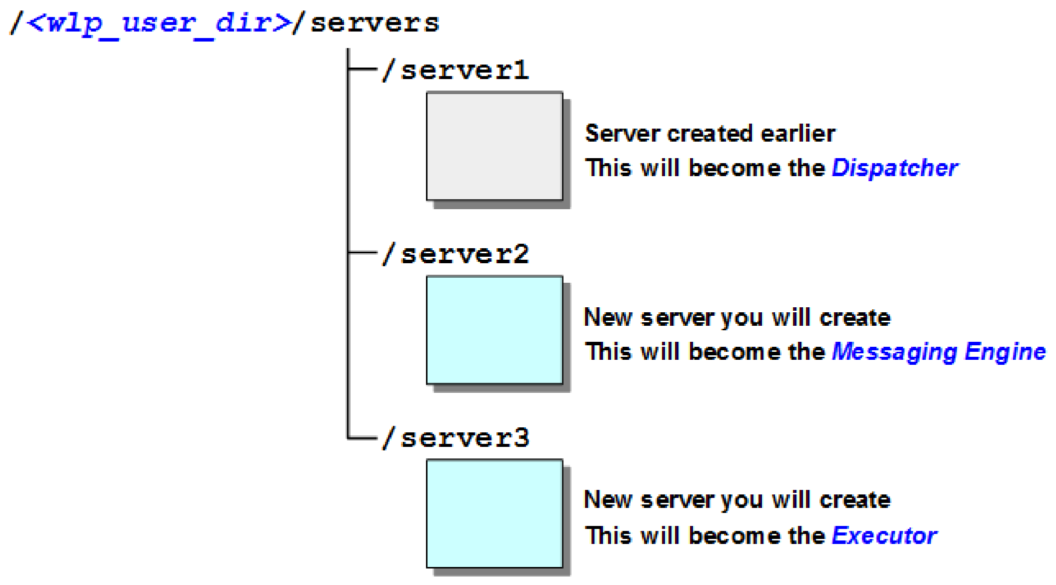
50 See "A note about undeliverable messages" on page 75 for more on undeliverable messages.

51 It involves setting up one Liberty Profile server as a network server so the other Liberty Server can access the Derby DB through a TCP port. Again, doable … just more involved then we wish to get into in this document. DB2 z/OS is easier.

The objective in this document is to help you see this work in the simplest way initially, so let's turn to what we'll illustrate and guide you to configure and use.

Design illustrated in this document

The following diagram illustrates what the step-by-step instructions will guide you to construct:


/<wlp_user_dir>/servers
/server1
Server created earlier
This will become the Dispatcher
/server2
New server you will create
This will become the Messaging Engine
/server3
New server you will create
This will become the Executor

Notes:

- By creating the servers under the same *<wlp_user_dir>* it makes it easier to start and operate under the same ID as the original server, whether started as a UNIX process or as a z/OS started task.
- We will illustrate the use of the built-in messaging function of Liberty Profile. This allows you to configure and use message queuing without involving MQ at this point in your testing. Later you can swap out the built-in messaging for MQ.
- We will guide you to create a separate server for the Messaging Engine[52]. While this *could* be in either the Dispatcher or Executor servers, making it separate keeps the configuration a little easier to follow and understand.

Create the server2 and server3 and validate basic start

Do the following:

☐ Use the same process you used to create the initial server:

  ○ Open a Telnet or OMVS session
  ○ Change directories to the `/bin` directory of your Liberty Profile installation
  ○ Export `JAVA_HOME` so it points to a valid 64-bit SDK for the system
  ○ Export `WLP_USER_DIR` so it points to the directory under which the first server is located. This is the directory immediately *above* `/servers/server1`, for example.
  ○ Create the second server: `./server create server2`
  ○ Create the third server: `./server create server3`

☐ Edit the `server.xml` for *each* new server and add `host="*"` to `httpEndpoint` section of the file.

52 Think of this is comparable to an MQ Queue Manager. It's where the queue is hosted.

- ☐ Edit the `server.xml` for each new server and update the `httpPort` and httpsPort so they are unique from `server1`, making sure the number you use does not conflict with other port usage on your system. For example:

	server1	**server2**	**server3**
httpPort	9080	9081	9082
httpsPort	9443	9444	9445

- ☐ Start each server and review the `messages.log` file to insure the server starts properly and the ports open:
  - `./server start server2`
  - `./server start server3`
- ☐ Stop both servers.

Configure the Messaging Engine (ME) server and start

The Messaging Engine is what implements the message queues, and what other servers access to put messages and retrieve messages. The queues themselves are held in files under the Liberty Profile server's `/messaging` directory[53].

Do the following:

- ☐ Edit the `server.xml` for `server2`. Add the lines highlighted below[54]:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
        <feature>wasJmsServer-1.0</feature>
    </featureManager>

    <wasJmsEndpoint
      host="*"
      id="InboundJmsEndpoint"
      wasJmsPort="7280"
      wasJmsSSLPort="7290"
      enabled="true">
    </wasJmsEndpoint>

    <messagingEngine>
      <queue id="batchLibertyQueue"
        forceReliability="ReliablePersistent"
        receiveAllowed="true"/>
    </messagingEngine>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="9081"
                  httpsPort="9444" />

</server>
```

- ☐ Save the file.
- ☐ Start the server.

53 The `/messaging` directory is created when a server configured to host the messaging engine starts.
54 See "A note about undeliverable messages" on page 75 for information on handling undeliverable messages.

- ☐ Look in the `messages.log` file for the server. You should see the following messages:

```
CWSIS1553I: The minimum reserved size in the configuration
information of the permanent store file is 20,971,520 bytes. The
maximum size is 209,715,200 bytes.

CWSIS1557I: The minimum reserved size in the configuration
information of the temporary store file is 20,971,520 bytes. The
maximum size is 209,715,200 bytes.

CWSID0108I: JMS server has started.

CWWKO0219I: TCP Channel JfapInboundTCPChannel has been started
and is now listening for requests on host *  (IPv4) port 7280.
```

  If you don't see those messages, then look for error messages that might provide a clue as to why the messaging engine did not start.

Configure the Dispatcher server and start

This is the server you created earlier and on which you performed the initial validation of JSR 352. To turn it into a Dispatcher involves a few updates to the `server.xml` file.

However, more work may be needed, depending on how you configured the JSR 352 batch persistence earlier[55]:

If "in-memory," then ...	you will need to configure and validate DB2. See next.
If Derby, then ...	you will need to configure and validate DB2. See next.
If DB2, then ...	you are ready to configure server as a Dispatcher. Go to "Configure server as Dispatcher" on page 46.

Configure DB2 as batch persistence

Do the following:

- ☐ Review "Database product (DB2 z/OS)" on page 18. That provides guidance on establishing the JSR 352 persistence tables in DB2 z/OS.
- ☐ Work with your z/OS DBA to get those tables created.
- ☐ Review "DB2 z/OS" on page 20. That provides guidance on on configuring server.xml for JDBC Type 4 to DB2.
- ☐ Update your `server.xml` with those definitions and remove any Derby definitions for JSR persistence, if those are present.
- ☐ Check `messages.log` for any obvious errors that might indicate some XML error or other definitional problem.
- ☐ Using the same invocation mechanism you used before, submit either SleepyBatchlet or BonusPayout and see if it completes successfully.
- ☐ Check the table on page 23 that provides information on how to make sure it used the persistence store you intended. Make sure it did in fact use DB2, and not in-memory or Derby.

Configure server as Dispatcher

With DB2 as the JSR 352 persistence location, you may move on and configure the first server as a Dispatcher. Do the following:

55 The multi-JVM support requires the Dispatcher and the Executors to share the same JobRepository persistence store. In-memory can't be shared, and sharing Derby is more involved than we'll get into in this document. We'll show DB2 here.

- ☐ Remove any applications from the `/dropins` directory for the server. This is not strictly required, but it eliminates any doubts about the Dispatcher running the applications locally.
- ☐ Add the following line to your `<featureManager>` section:

```
<feature>wasJmsClient-2.0</feature>
```

- ☐ Now add the following XML to the `server.xml`:

```
<batchJmsDispatcher
  connectionFactoryRef="batchConnectionFactory"
  queueRef="batchJobSubmissionQueue" />

<jmsConnectionFactory id="batchConnectionFactory"
  jndiName="jms/batch/connectionFactory">
 <properties.wasJms
    remoteServerAddress="<host>:7280:BootstrapBasicMessaging">
 </properties.wasJms>
</jmsConnectionFactory>

<jmsQueue id="batchJobSubmissionQueue"
  jndiName="jms/batch/jobSubmissionQueue">
  <properties.wasJms deliveryMode="Persistent"
    queuename="batchLibertyQueue">
  </properties.wasJms>
</jmsQueue>
```

  Where *`<host>`* is the host name where the second server – the Messaging Engine server – is running. That port (`7280`) is what was defined as the Messaging Engine listener port for that server.

- ☐ Look in the `messages.log` file for the indication key features are installed:

```
CWWKF0012I: The server installed the following features:
                             [... mdb-3.2, ... wasJmsClient-2.0].
```

Configure the Executor server and start

The Executor server will listen on the queue hosted by the Messaging Engine server, and pick up messages that match the configured selector criteria. It will then run the job.

Do the following:

- ☐ Put the SleepyBatchlet WAR file in the `/dropins` directory of `server3`. The server will need the batch application deployed to be able to run it.
- ☐ Update the `<featureManager>` section of `server.xml` so it has what's needed to run batch applications and serve as an Executor:

```
<featureManager>
    ~~<feature>jsp-2.2</feature>~~
    <feature>servlet-3.1</feature>
    <feature>batch-1.0</feature>
    <feature>batchManagement-1.0</feature>
    <feature>wasJmsClient-2.0</feature>
</featureManager>
```

- ☐ Add *all* of the following XML to designate the server as an Executor as well as point to the Messaging Engine and queue:

```
<batchJmsExecutor
  activationSpecRef="batchActivationSpec"
  queueRef="batchRequestsQueue"/>

<jmsActivationSpec id="batchActivationSpec" maxEndpoints="5">
```

```
  <properties.wasJms destinationRef="batchRequestsQueue"
   destinationType="javax.jms.Queue"
   messageSelector="com_ibm_ws_batch_applicationName =
                             'SleepyBatchletSample-1.0'"
   remoteServerAddress="<host>:7280:BootstrapBasicMessaging">
  </properties.wasJms>
</jmsActivationSpec>

<jmsQueue id="batchRequestsQueue"
  jndiName="jms/batch/jobSubmissionQueue">
  <properties.wasJms
    queueName="batchLibertyQueue"
    deliveryMode="Persistent"/>
</jmsQueue>
```

Where `<host>` is the host on which the Messaging Engine server is running.

Note the `messageSelector=` line. As coded here, that tells the Executor to look for and pick up *only* job submission messages that match the application name for SleepyBatchlet. It will not pick up messages for other applications as coded here. Later we will show how to add an OR operator to have it listen for multiple applications.

- ☐ Configure this server to use DB2 `<batchPersistence>` just like the Dispatcher server used. That's key – the Dispatcher and Executors must share the same batch persistence store. Copy the XML for batch persistence and DB2 JDBC from `server1` to this `server3`.
- ☐ After you've saved the file (or restarted the server), look for the following messages in the messages.log file:

```
CWSIV0777I: A connection to messaging engine defaultME for
destination batchLibertyQueue on bus defaultBus has been
successfully created.

CWSIV0556I: Connection to the Messaging Engine was successful.
The message-driven bean with activation specification
batchActivationSpec will now be able to receive the messages from
destination batchLibertyQueue.

J2CA8801I: The message endpoint for activation specification
batchActivationSpec and message driven bean application
JBatchListenerApp#JBatchListenerModule#JBatchListenerComp is
activated.
```

That indicates the Executor server has connected to the Messaging Engine and is listening on the queue. If you don't see that message, then something is not working properly. Look for other messages indicating some problem, and make sure the XML copied in is as shown above.

Submit job and see it dispatched and run in Executor

At this point you should have your environment established – a Messaging Engine to host the queue; a Dispatcher that will take the job submission and put a message on the queue; and an Executor that listens on the queue and picks up the job submission message and runs the batch application.

Job submission to the Dispatcher is *exactly* as you did earlier – using either the REST interface directly, the `batchManager` command line client or the `batchManagerZos` command line client.

Do the following:

- ☐ Make sure all three servers are up.

- ☐ Choose the method of job submission you wish to use – REST client, `batchManager` or `batchManagerZos`.
- ☐ Submit the SleepyBatchlet job just as you did before. You should see the Dispatcher server respond back that the job has started. For example, using `batchManagerZos` it would look like this:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
                           --applicationName=SleepyBatchletSample-1.0
                             --jobXMLName=sleepy-batchlet.xml --wait
```

- ☐ Look in the Dispatcher server's `messages.log` file. You will *not* see any output from the SleepyBatchlet application. That's because it's not running there.
- ☐ Look in the Executor server's `messages.log` file. You should see the output messages from SleepyBatchlet there.
- ☐ If you used the `--wait` parameter the submission client will wait for the job to complete. If you go back to the Telnet window from which you submitted the job, you should see a message indicating the job has completed.

Modify messageSelector so Executor runs multiple applications

If you look in the Executor's `<jmsActivationSpec>` section of `server.xml`, you'll see the following:

```
messageSelector="com_ibm_ws_batch_applicationName =
                                   'SleepyBatchletSample-1.0'"
```

That tells the Executor to pick up *only* the SleepyBatchlet job submission. Any other job submission message with another application name will not be picked up.

You can tell the `messageSelector` to look for multiple applications. Do the following:

- ☐ Modify the messageSelector element so it reads like this:

```
messageSelector="com_ibm_ws_batch_applicationName =
                                   'SleepyBatchletSample-1.0'
                OR com_ibm_ws_batch_applicationName =
                                   'BonusPayout-1.0'"
```

Note: pay attention to the opening double-quote and closing double-quote. Before the closing double-quote was after the SleepyBatchlet name, now it is after the BonusPayout name.

Note also the blank spaces before and after the equal sign for each application name.

The message selector may be more than just the application name. See the KnowledgeCenter article with unique search string `twlp_batch_multiJVMembed` for more.

- ☐ Save the file.
- ☐ See "Using the BonusPayout chunk application sample" on page 30 for information on setting up and using that sample application. If not already done, perform the steps necessary to create the table and configure the JDBC for the table. Also, deploy the application WAR file into the `/dropins` directory.
- ☐ Submit through the Dispatcher the SleepyBatchlet sample, just as you did earlier. You should see this run in the Executor.
- ☐ Submit through the Dispatcher the BonusPayout sample, using a command line such as the following:

```
./batchManagerZos submit –batchManager=LIBERTY+BATCH+MANAGER
```

```
                                  --applicationName=BonusPayout-1.0
                              --jobXMLName=SimpleBonusPayoutJob.xml
                                   --jobParameter=dsJNDI=jdbc/bonus
                       --jobParameter=tableName=BONUS.ACCOUNT --wait
```

- ☐ The BonusPayout sample does not leave much evidence in the `messages.log` file, but you should see `jobLog` for it[56].

If both SleepyBatchlet and BonusPayout both ran, that indicates the OR operator on the two message selector clauses worked.

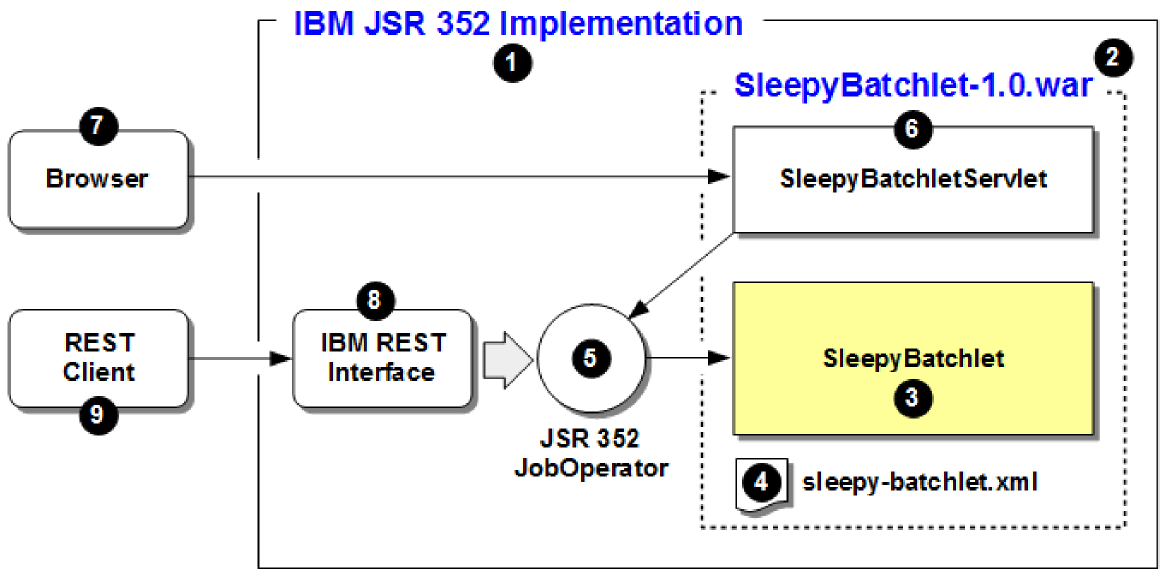
56 See "Job logging" on page 24.

Review of the JSR 352 Sample Applications

The SleepyBatchlet sample

The SleepyBatchlet sample is a very simple batchlet step that "sleeps" for a specified period of time, then it finishes. There is no input or output requirements.

Inside the WAR file you'll find two class files – one is the batchlet step itself; the other is a servlet that acts as an code to initiate the job using the JSR-352 JobOperator interface. Another way to initiate the job is using the REST interface we saw earlier (page 25). The SleepyBatchlet sample works either way:



Notes:

1. The "IBM JSR 352 Implementation" is what we're exploring in this Getting Started Guide. It is what runs inside Liberty Profile.
2. The sample application is packaged as a WAR file. It is this WAR file you placed in the /dropins directory of the Liberty Profile server.
3. The JSR 352 batchlet job step implementation is contained within the `SleepyBatchlet.class` file. We will look at the source Java to this later in this section.
4. The Job Specification Language (JSL) file that describes the JSR 352 batch job.
5. The JSR 352 JobOperator interface. This is needed to start and control a JSR 352 batch job.
6. Some function is required to invoke the JobOperator interface. The SleepyBatchlet sample supplies its own servlet for this role.

   IBM's JSR 352 implementation also provides a functional interface to the JobOperator. It is item ❽ in the picture above – the REST Interface[57]. Therefore, the Servlet is not really needed. However, the sample provides it, and the SleepyBatchlet job step may be invoked through either the Servlet or the REST interface.

7. A browser is all that's needed to invoke the Servlet, which then works against the JobOperator interface, which starts the SleepyBatchlet job.
8. The REST Interface is IBM's implementation of a functional implementation on the front of the JobOperator interface. On z/OS a comparable interface is implemented using WOLA, with the client for that interface being the `batchManagerZos` command line client.
9. Any REST client, including the IBM-supplied `batchManager` client, can be used to start the batch job through the REST Interface.

57 Or, on z/OS, the REST interface *and* the WOLA interface used by the `batchManagerZos` command line client.

Sample and source on Github

The sample can be found on GitHub at the following location:

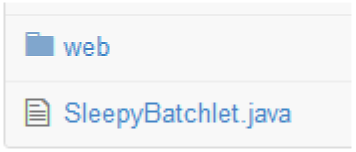
https://github.com/WASdev/sample.batch.sleepybatchlet

The sample WAR does not contain the source Java, but the GitHub site location has the source Java for review and download:

→ *URL for SleepyBatchlet*

→ *src/main*

→ *java/com/ibm/ws/jbatch/sample/sleepybatchlet*



The "web" folder contains the source for the servlet.

SleepyBatchlet source review

SleepyBatchlet

Going from top to bottom in the source, we see:

```
import javax.batch.api.AbstractBatchlet;
import javax.batch.api.BatchProperty;
import javax.inject.Inject;
```

The first two are related to JSR 352 Java batch, and the third is to identify injectible fields.

```
private boolean stopRequested = false;
```

This is used in the `process()` method to determine if the sleep loop is to be stopped. By default the value is `false` (do not stop; keep going). The `stop()` method has code to set this to `true` (see below) which is how the batchlet can be made to stop before normal completion.

```
@Inject
@BatchProperty(name = "sleep.time.seconds")
String sleepTimeSecondsProperty;
```

The property `sleep.time.seconds` is one that may be passed in on job submission. If it is passed in, then the string `sleepTimeSecondsProperty` is set to the passed-in property.

```
private int sleepTime_s = 15;
```

The integer variable `sleepTime_s` is set to 15 seconds just in case a sleep time property is not passed in at job submission. You'll see these used in the `process()` method.

```
if (sleepTimeSecondsProperty != null) {
        sleepTime_s = Integer.parseInt(sleepTimeSecondsProperty);
}
```

This is found in the `process()` method. If the sleep time property was passed in at job submission, then the sleep time is set to that value. Otherwise, the private default value of `sleepTime_s` is in effect.

```
for (i = 0; i < sleepTime_s && !stopRequested; ++i) {
            log("process", "[" + i + "] sleeping for a second...");
            Thread.sleep(1 * 1000);
}
```

This `for` loop is the heart of the SleepyBatchlet job step. This will continue to loop until the number of seconds is up or the `stopRequested` boolean is seen as not `true`. That boolean is set to `false` by default (see above), and is only set to `true` if the `stop()` method is invoked (see next).

```
public void stop() throws Exception {
    log("stop:", "");
    stopRequested = true;
}
```

The `stop()` method's role is to set the `stopRequested` boolean to `true` when invoked. Then, as we saw in the sleep loop, the for loop logic checks to see the value of this boolean. When this method is invoked, the boolean is set to `true`, and on the next iteration through the loop the logic there will see value `true` and stop processing.

Note: we bring this to your attention for this reason – the JSR 352 specification calls for batchlet steps to have a `stop()` method, but the default implementation doesn't actually do anything to stop the batchlet. Whoever designs and writes the batchlet job step must implement logic to stop the batchlet when the `stop()` method is invoked. In this sample we see that was done with the `stopRequested` boolean and a check in the `for` loop in `process()`.

That's it. The batchlet is a relatively simple loop to sleep one second for as many seconds as passed in on the `sleep.time.seconds` property at job submission *or* the default 15 seconds. The `stop()` method sets the boolean `stopRequested`, which is checked in the `process()` method sleep loop.

SleepyBatchletServlet source review

As noted earlier, the purpose of the SleepyBatchletServlet is to provide a browser interface for job submission and job control.

Note: IBM's JSR 352 implementation in Liberty Profile provides a REST interface (see “REST Interface” on page 25 for more 27), as well as a command line client (see “batchManager utility” on page for more) and a native command line utility on z/OS (see “Use batchManagerZos” on page 39 for more).

This servlet is *not required* to use SleepyBatchlet in the Liberty Profile implementation of JSR 352. The REST interface can be used to submit and control the batchlet job. The servlet is in the sample because it makes using the sample very easy – use of the servlet does not require `batchManagement-1.0` to be coded in the `server.xml` (the REST interface does), and it does not require the security setup `batchManagement-1.0` requires.

Let's take a look at what the servlet does. Starting at the top and working to the bottom:

```
import javax.batch.operations.JobOperator;
import javax.batch.runtime.BatchRuntime;
import javax.batch.runtime.JobExecution;
import javax.batch.runtime.JobInstance;
```

These provide the imports to use the JobOperator interface of JSR 352.

```
String action = request.getParameter("action");

      if ("start".equalsIgnoreCase(action)) {
          start(request, response);
      } else if ("status".equalsIgnoreCase(action)) {
          status(request, response);
      } else if ("stop".equalsIgnoreCase(action)) {
          stop(request, response);
      } else if ("restart".equalsIgnoreCase(action)) {
          restart(request, response);
      } else {
          throw new IOException("action not recognized: " + action );
      }
```

This bit of code is extracting the `&action=` parameter from the URL that was used to invoke the servlet. For actions are supported by this servlet – `start`, `status`, `stop` and `restart`. If something other than one of those four is provided, an exception is thrown.

From the `start()` method:

```
JobOperator jobOperator = BatchRuntime.getJobOperator();
long execId = jobOperator.start("sleepy-batchlet",
                    getJobParameters(request, "sleep.time.seconds"));
JobInstance jobInstance = jobOperator.getJobInstance(execId);
JobExecution jobExecution = jobOperator.getJobExecution(execId);
```

The first line gets a reference to the `jobOperator` object, then it uses the `jobOperator.start()` method to invoke the job.

The second line submits the job and gets an identifier for the job. The string "sleepy-batchlet" refers to the Job Specification Language file for the job, which is packaged in the sample application's WAR file. The sleep time can be passed in as a parameter on the URL in the form of `&sleep.time.seconds=`*`xx`*, and if present that value is passed to the job.

The final two lines get the Job Instance ID and Job Execution ID for the submitted job.

From the `status()` method:

```
long execId = getLongParm(request, "executionId");
JobOperator jobOperator = BatchRuntime.getJobOperator();
JobInstance jobInstance = jobOperator.getJobInstance(execId);
List<JobExecution> jobExecutions = jobOperator.getJobExecutions(jobInstance);
```

When `?action=status&executionID=`*`xx`* is on the URL, this method is called. The first line puts the execution ID parameter from the URL into `execID`.

A reference to the jobOperator and jobInstance is retrieved, then the status for the job, based on the execution ID, is requested.

From the `stop()` method:

```
long execId = getLongParm(request, "executionId");
BatchRuntime.getJobOperator().stop(execId);
```

When `?action=stop&executionID=`*`xx`* is on the URL, this method is called. The first line puts the execution ID parameter from the URL into `execID`.

Then a stop request is issued[58] based on the execution ID.

From the `restart()` method:

```
long execId = getLongParm(request, "executionId");
long newExecId = BatchRuntime.getJobOperator().restart(execId,
                            getJobParameters(request, "sleep.time.seconds"));
```

When `?action=stop&executionID=`*`xx`* is on the URL, this method is called. Then the execution ID is used to restart the job. This results in a new execution ID being created by the batch runtime, and this value is held in `newExecID`.

58 Recall the `stop()` method in the SleepyBatchlet code … it simply sets the boolean `stopRequsted` to `true`. The for loop in the `process()` method has a check for the state of the boolean. When `true`, then the loop is exited and the job step stops.

The rest of the servlet is standard servlet error handling code. The elements of the servlet that handled JSR 352 interactions are shown above.

Job Specification Language (JSL) file

The Job Specification Language (JSL) file is a required element of the JSR 352 specification. It is an XML file, and it is used to provide details about the job and the job steps contained there.

The JSR 352 specification indicates that the JSL be packaged with the batch application under the `\classes\META-INF` directory.

The JSL for SleepyBatchlet is relatively simple:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<job id="sleepy-batchlet" xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">
   <step id="step1">  1
      2 <batchlet ref="com.ibm.ws.jbatch.sample.sleepybatchlet.SleepyBatchlet" >
           <properties>
               <property name="sleep.time.seconds"  3
                   value="#{jobParameters['sleep.time.seconds']}" />  4
           </properties>
       </batchlet>
   </step>
</job>
```

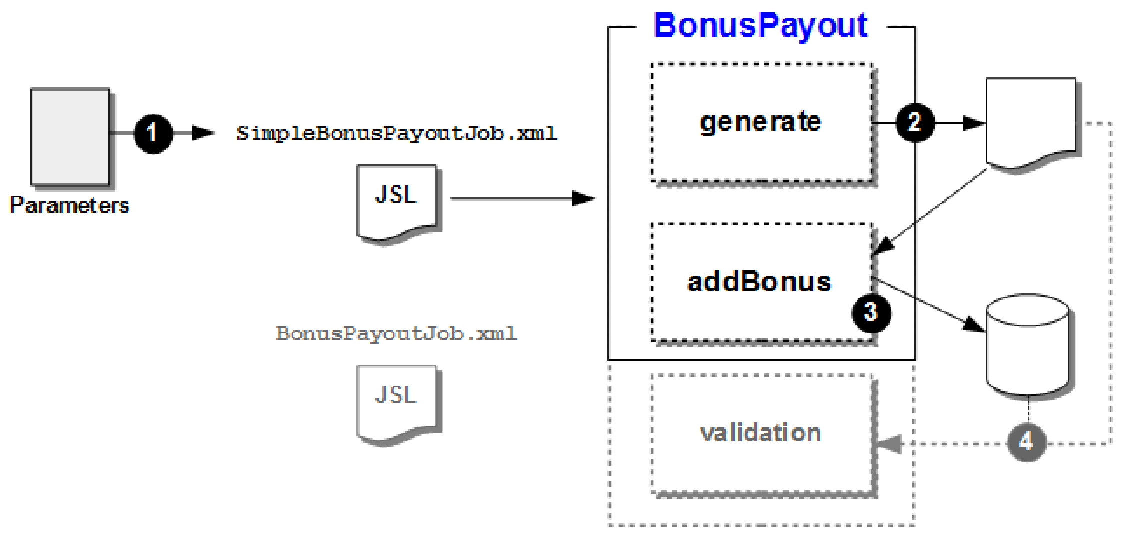
Notes:

1. The job has one step
2. The step is defined as “batchlet” and the class file implementation for the batchlet is specified. That class file is included in the WAR file.
3. The step has one property – the number of seconds the batchlet will “sleep”.
4. The single step property can be passed in as a parameter at time of job submission

The BonusPayout sample

Earlier (“Using the BonusPayout chunk application sample” page 30) we guided you through using the BonusPayout sample. In this section we'll take a little closer look at what went on inside that sample application.

Back in that section we showed you this picture:



At a high-level, that batch job creates a file (“generate” step), then reads from the file and updates a relational database table (“addBonus” step). The third step (“validation”) is optional, and is run or not run based on which packaged Job Specification Language (JSL) file you point to at submission. That third step compares the file against the table to validate the first two steps worked as designed.

In this section we'll step back a bit from line-by-line commentary about the code in the sample. It is largely standard Java … input and output streams, and JDBC activity.

Sample and source on Github

The sample can be found on GitHub at the following location:

`https://github.com/WASdev/sample.batch.bonuspayout`

You need to drill down a bit to get at the source documents:

→ *URL for BonusPayout*

  → *BonusPayout link*

    → *src/main link*

      → *java/com/ibm/websphere/samples/batch link*

        → *artifacts link*

The source files we'll explore here will be:

Java Source	*Step Used and Function Provided*
`GenerateDataBatchlet`	*First step … "generate"* This is a batchlet step. This Java creates the data in the UNIX file that will be read by the second step.
`GeneratedCSVReader`	*Second step … "addBonus"* This is a chunk step, which means it must implement ItemReader, ItemProcessor and ItemWriter. That's what these three Java artifacts provide. *GenerateCSVReader* – reads from the file generated in the first step of this job. *BonusCreditProcessor* – adds the fixed bonus amount to each record found in the file. *AccountJDBCWriter* – inserts the updated account records into the relational database table.
`BonusCreditProcessor`	
`AccountJDBCWriter`	

Job Specification Language (JSL) file

The BonusPayout sample contains two packaged JSL files:

- `SimpleBonusPayoutJob.xml`
- `BonusPayoutJob.xml`

The first is what we will illustrate here. It invokes the first two steps only. The second JSL file is used to invoke all three steps. The JSL is below, with notes following:

```
<?xml version="1.0" encoding="UTF-8"?>

<job id="SimpleBonusPayoutJob" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    :
   <properties> 1
 2  <property name="numRecords" value="#{jobParameters['numRecords']}?:1000;" />
 3  <property name="chunkSize" value="#{jobParameters['chunkSize']}?:100;" />
 4  <property name="dsJNDI" value="#{jobParameters['dsJNDI']}?:java:comp/env/jdbc/BonusPayoutDS;" />
 5  <property name="bonusAmount" value="#{jobParameters['bonusAmount']}?:100;" />
 6  <property name="tableName" value="#{jobParameters['tableName']}?:BONUSPAYOUT.ACCOUNT;" />

    <!-- These next two will be accessed via the JobContext, rather than via injected
       through @Inject @BatchProperty injection -->
```

```
7  <property name="fileEncoding" value="#{jobParameters['fileEncoding']}" />
8  <property name="generateFileNameRoot" value="#{jobParameters['generateFileNameRoot']}" />
   </properties>

   <!-- Generate random account-like data in CSV format into a text file -->

   <step id="generate" next="addBonus">  9
      <batchlet
         ref="com.ibm.websphere.samples.batch.artifacts.GenerateDataBatchlet">  10
         <properties>
            <property name="numRecords" value="#{jobProperties['numRecords']}" />  11
         </properties>
      </batchlet>
   </step>

   <!-- Increment each account with configurable 'bonusAmount' (integer). -->
   <step id="addBonus">  12
      <chunk item-count="#{jobProperties['chunkSize']}">  13
         <reader ref="com.ibm.websphere.samples.batch.artifacts.GeneratedCSVReader"/>  14
         <processor
            ref="com.ibm.websphere.samples.batch.artifacts.BonusCreditProcessor">  15
            <properties>
               <property name="bonusAmount" value="#{jobProperties['bonusAmount']}" />  16
            </properties>
         </processor>
         <writer
            ref="com.ibm.websphere.samples.batch.artifacts.AccountJDBCWriter">  17
            <properties>
               <property name="dsJNDI" value="#{jobProperties['dsJNDI']}" />  18
               <property name="tableName" value="#{jobProperties['tableName']}" />  19
            </properties>
         </writer>
      </chunk>
   </step>

</job>
```

Notes:

1. The start of the job properties section. These properties may take the default as set in the JSL file, or be overridden at time of submission by passing in the properties as parameters.
2. The `numRecords` property determines how many data records will be created in the first step, and written to the database in the second step. It defaults to 1000.
3. The `chunkSize` property determines how frequently the JSR 352 container will process a commit during write operations to the database. It defaults to 100.
4. The `dsJNDI` property is used by the application to perform a lookup of the JDBC data source for the relational database it will access. The default is shown in the diagram above. You may take the default or override it with a job parameter. The value that takes effect must match a properly defined JDBC data source JNDI in the `server.xml` for the server.
5. The `bonusAmount` property determines how much is added to the balance of each data record during processing of the second step.
6. The `tableName` property determines what database table to access.
7. The `fileEncoding` property is used to set the codepage for the file created in the first step. This defaults to ASCII.
8. The `generateFileNameRoot` property is used to determine where the data file will be created and what name prefix it will carry. By default the file will be created in the

server's root directory (where `server.xml` resides) and will have a name of `.#.csv`, where `#` is the job execution ID.

9. The start of the “generate” job step. This is defined as a “batchlet” job step. This job step creates the data file and populates it with the number of data records indicated by `numRecords`. The record looks something like this:

   ```
   0,562,CHK
   1,287,CHK
   2,471,CHK
     :
   998,904,CHK
   999,654,CHK
   ```

   Where the first field is the account number, the second is the account balance, and the third is the account type.
10. The pointer to the Java class that implements the batchlet job step.
11. The job property used by this job step, which is just `numRecords`.
12. The start of the “addBonus” job step.
13. This is defined as a “chunk” job step, with the commit intervale (“chunk size”) specified by the job property `chunkSize`.
14. The Java class that implements the ItemReader for this chunk step. This class reads a record from the file created in the “generate” job step.
15. The Java class that implements the ItemProcessor for this chunk step. This class adds the bonus amount to the balance for each record processed.
16. The job property specifying the bonus amount to be added to each record.
17. The Java class that implements the ItemWriter for this chunk step.
18. The job property for the JNDI of the JDBC data source for access to the relational database.
19. The job property for the relational table name to be accessed.

BonusPayout source for key program elements

GenerateDataBatchlet

```
/*
 * Copyright 2014 International Business Machines Corp.
 *
 * See the NOTICE file distributed with this work for additional information
 * regarding copyright ownership. Licensed under the Apache License,
 * Version 2.0 (the "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.ibm.websphere.samples.batch.artifacts;

import java.io.BufferedWriter;
import java.util.Random;
import java.util.logging.Logger;

import javax.batch.api.BatchProperty;
```

```
import javax.batch.api.Batchlet;
import javax.batch.runtime.context.JobContext;
import javax.inject.Inject;

import com.ibm.websphere.samples.batch.beans.AccountType;
import com.ibm.websphere.samples.batch.beans.CheckingAccountType;
import com.ibm.websphere.samples.batch.beans.PriorityAccount;
import com.ibm.websphere.samples.batch.util.BonusPayoutUtils;
import com.ibm.websphere.samples.batch.util.BonusPayoutConstants;

/**
 * Generate some random data, then write in CSV format into a text file.
 */
public class GenerateDataBatchlet implements Batchlet, BonusPayoutConstants {

    private final static Logger logger = Logger.getLogger(BONUS_PAYOUT_LOGGER);

    /**
     * How many records to write.
     */
    @Inject
    @BatchProperty(name = "numRecords")
    private String numRecordsStr;

    /**
     * File to write generated data to.
     */
    @Inject
    @BatchProperty
    private String generateFileNameRoot;

    @Inject
    private JobContext jobCtx;

    /**
     * Set to whatever you want, could make a parameter/property.
     */
    private final int maxAccountValue = 1000;

    /*
     * For CDI version of sample this will be injectable.
     */
    private AccountType acctType = new CheckingAccountType();

    /*
     * Included for CDI version of sample.
     */
    @Inject
    public void setAccountType(@PriorityAccount AccountType acctType) {
        this.acctType = acctType;
    }

    private boolean stopped = false;

    private BufferedWriter writer = null;

    @Override
    public String process() throws Exception {

        writer = new BonusPayoutUtils(jobCtx).openCurrentInstanceStreamWriter();

        String accountCode = acctType.getAccountCode();

        logger.info("In GenerateDataBatchlet, using account code = " + accountCode);

        int numRecords = Integer.parseInt(numRecordsStr);
        for (int i = 0; i < numRecords; i++) {

            StringBuilder line = new StringBuilder();

            // 1. Write record number
            line.append(i).append(',');

            // 2. Write random value
            line.append(new Random().nextInt(maxAccountValue)).append(',');

            // 3. Write account code
            line.append(accountCode);

            writer.write(line.toString());
            writer.newLine();

            if (stopped) {
                logger.info("Aborting GenerateDataBatchlet since a stop was received");
                writer.close();
                return null;
            }
        }
        writer.close();

        return null;
    }

    @Override
    public void stop() {
        stopped = true;
    }

}
```

GeneratedCSVReader

```
/*
 * Copyright 2014 International Business Machines Corp.
 *
 * See the NOTICE file distributed with this work for additional information
 * regarding copyright ownership. Licensed under the Apache License,
 * Version 2.0 (the "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.ibm.websphere.samples.batch.artifacts;

import java.io.BufferedReader;
import java.io.Serializable;
import java.util.logging.Logger;

import javax.batch.api.chunk.ItemReader;
import javax.batch.runtime.context.JobContext;
import javax.inject.Inject;

import com.ibm.websphere.samples.batch.beans.AccountDataObject;
import com.ibm.websphere.samples.batch.util.BonusPayoutUtils;
import com.ibm.websphere.samples.batch.util.BonusPayoutConstants;

/**
```

```
 * Parses a line of the CSV file into an AccountDataObject.
 */
public class GeneratedCSVReader implements ItemReader, BonusPayoutConstants {

    private final static Logger logger = Logger.getLogger(BONUS_PAYOUT_LOGGER);

    @Inject
    private JobContext jobCtx;

    // Next line to read, 0-indexed.
    int recordNumber = 0;

    private BufferedReader reader = null;

    @Override
    public Object readItem() throws Exception {
        String line = reader.readLine();
        if (line == null) {
            logger.fine("End of stream reached in " + this.getClass());
            return null;
        } else {
            AccountDataObject acct = BonusPayoutUtils.parseLine(line);
            recordNumber++;
            return acct;
        }
    }

    @Override
    public void open(Serializable checkpoint) throws Exception {
        if (checkpoint != null) {
            recordNumber = (Integer) checkpoint;
        }

        reader = new BonusPayoutUtils(jobCtx).openCurrentInstanceStreamReader();

        // Advance cursor (not worrying  much about performance)
        for (int i = 0; i < recordNumber; i++) {
            reader.readLine();
        }
    }

    @Override
    public void close() throws Exception {
        reader.close();
    }

    @Override
    public Serializable checkpointInfo() throws Exception {
        return recordNumber;
    }
}
```

BonusCreditProcessor

```
/*
 * Copyright 2014 International Business Machines Corp.
 *
 * See the NOTICE file distributed with this work for additional information
 * regarding copyright ownership. Licensed under the Apache License,
 * Version 2.0 (the "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.ibm.websphere.samples.batch.artifacts;

import javax.batch.api.BatchProperty;
import javax.batch.api.chunk.ItemProcessor;
import javax.inject.Inject;

import com.ibm.websphere.samples.batch.beans.AccountDataObject;
import com.ibm.websphere.samples.batch.util.BonusPayoutConstants;

/**
 * Credit each balance with a 'bonus' amount.
 */
public class BonusCreditProcessor implements ItemProcessor, BonusPayoutConstants {

    @Inject
    @BatchProperty(name = "bonusAmount")
    String bonusAmountStr;

    Integer bonusAmount = null;

    @Override
    public Object processItem(Object item) throws Exception {
        AccountDataObject ado = (AccountDataObject) item;

        int newBalance = ado.getBalance();
        ado.setBalance(newBalance + getBonusAmount());
        return ado;
    }

    private int getBonusAmount() {
        if (bonusAmount == null) {
            bonusAmount = Integer.parseInt(bonusAmountStr);
        }

        return bonusAmount;
    }

}
```

AccountJDBCWriter

```
/*
 * Copyright 2014 International Business Machines Corp.
 *
 * See the NOTICE file distributed with this work for additional information
 * regarding copyright ownership. Licensed under the Apache License,
 * Version 2.0 (the "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.ibm.websphere.samples.batch.artifacts;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.util.List;
import java.util.logging.Logger;
import javax.batch.api.BatchProperty;
import javax.batch.api.chunk.AbstractItemWriter;
import javax.batch.api.chunk.ItemWriter;
import javax.batch.runtime.context.JobContext;
import javax.inject.Inject;
import javax.sql.DataSource;

import com.ibm.websphere.samples.batch.beans.AccountDataObject;
import com.ibm.websphere.samples.batch.util.BonusPayoutUtils;
import com.ibm.websphere.samples.batch.util.BonusPayoutConstants;

/**
 * Loops through the items list building and finally executing a batch insert.
 *
 * Follow get-use-close pattern with JDBC Connection.
 */
public class AccountJDBCWriter extends AbstractItemWriter implements ItemWriter,
BonusPayoutConstants {

    private final static Logger logger = Logger.getLogger(BONUS_PAYOUT_LOGGER);

    @Inject
    @BatchProperty
    private String dsJNDI;

    @Inject
    @BatchProperty
    private String tableName;

    @Inject
    private JobContext jobCtx;

    private DataSource ds = null;

    @Override
    public void open(Serializable checkpoint) throws Exception {
        ds = BonusPayoutUtils.lookupDataSource(dsJNDI);
        BonusPayoutUtils.validateTableName(tableName);
    }

    @Override
    public void writeItems(List<Object> items) throws Exception {
        Connection conn = ds.getConnection();
        String sql = "INSERT INTO " + tableName + " VALUES (?,?,?,?)";
        PreparedStatement ps = conn.prepareStatement(sql);
        for (Object obj : items) {
            AccountDataObject ado = AccountDataObject.class.cast(obj);
            ps.setInt(1, ado.getAccountNumber());
            ps.setInt(2, ado.getBalance());
            ps.setLong(3, jobCtx.getInstanceId());
            ps.setString(4, ado.getAccountCode());
            ps.addBatch();
        }
        logger.fine("Adding: " + items.size() + " items to table name: " + tableName + " via
batch update");
        ps.executeBatch();
        logger.fine("Executed batch update.");
        ps.clearBatch();
        ps.close();
        conn.close();
    }

}
```

Miscellaneous Information

Installing the JSR 352 features into Liberty Profile

Liberty Profile uses *features* as composable "building blocks" of function. When Liberty Profile is installed using IBM Installation Manager, by default only a subset of the total features will be installed. The features you need for IBM's JSR 352 implementation are *not* included by default.

The information provided here illustrates how to determine what's currently installed, and how to install the features needed for the exercises in this document.

Determine the currently installed features

In the `/bin` directory of the product install file system there is a utility called `productInfo`. That can be used to list out the currently installed features.

Do the following:

- ☐ Telnet into the system (or use OMVS). Login with your IM admin ID.
- ☐ Change to the `/bin` directory of the product install.
- ☐ Issue the command:

  `./productInfo featureInfo`

  It will produce a listing of features, such as:

```
appSecurity-1.0 [1.1.0]
appSecurity-2.0 [1.0.0]
beanValidation-1.0 [1.0.0]
blueprint-1.0 [1.0.0]
 :
webProfile-6.0 [6.0.0]
zosSecurity-1.0 [1.0.0]
zosTransaction-1.0 [1.0.0]
zosWlm-1.0 [1.0.0]
```

- ☐ Look for the following features (which will likely ***not*** be there):

`batch-1.0` `batchManagement-1.0`	These are the key features for the JSR 352 support.

`jdbc-4.1`
`jca-1.6`
`servlet-3.1`
`zosLocalAdapters-1.0`
	These are features that are also required. They will be installed automatically when you install the two batch features.

- ☐ If those features are present, then you do not need to do anything further. Return to the step-by-step instructions earlier in the document and proceed.

Install the features you need

To install the required features, you use the `installUtility` function, which is located in the `/bin` directory of the product install file system.

Note: you may need to work with your installation person on this process. Installing features requires the file system to be mounted R/W, and yours may not be. Further, depending on whether your system has internet access, you may need to point `installUtility` at files in a local repository.

Do the following:

- ☐ Make sure the file system that contains the Liberty Profile product install is mounted as READ/WRITE. Installing features implies an update to the file system.

- ☐ If you do *not* have network access to the IBM hosted respository, then see "If your z/OS LPAR does not have internet access" on page 66. What follows below assumes you *do* have network access for installing features from the IBM cloud.
- ☐ In Telnet (or OMVS) session, change to the `/bin` directory of the product install.
- ☐ Install the `batch-1.0` feature:

```
./installUtility install batch-1.0 --acceptLicense
```

- ☐ You should see *something* like this:

```
The --acceptLicense argument was found. This indicates that you have
accepted the terms of the license agreement.
Step 1 of 11: Starting installation...
Step 2 of 11: Checking features...
Step 3 of 11: Resolving remote features. This process might take
several minutes to complete.
Step 4 of 11: Downloading servlet-3.1...
Step 5 of 11: Downloading jdbc-4.1...
Step 6 of 11: Downloading batch-1.0...
Step 7 of 11: Installing servlet-3.1...
Step 8 of 11: Installing jdbc-4.1...
Step 9 of 11: Installing batch-1.0...
Step 10 of 11: Cleaning up temporary files...
Step 11 of 11: Installation completed
CWWKF1017I: One or more features installed successfully: [servlet-
3.1, jdbc-4.1, batch-1.0].
```

The `batch-1.0` feature requires `servlet-3.1` and `jdbc-4.1`. The `featureManager` utility understands this and installs the related features.

- ☐ Do the same for the `batchManagement-1.0` feature.
- ☐ Do the same for the `zosLocalAdapters-1.0` feature.

This is needed for the `batchManagerZos` utility, which uses WOLA, which is what this feature provides. This will install `jca-1.6` as well.

Verify feature installation

This involves re-issuing the productInfo command to check the features that are installed.

- ☐ Issue the command:

```
./productInfo featureInfo
```

- ☐ Look for the following features (which now *should* be there):

`batch-1.0` `batchManagement-1.0`	These are the key features for the JSR 352 support.

`jdbc-4.1`
`jca-1.6`
`servlet-3.1`
`zosLocalAdapters-1.0`
	Features in support of the JSR 352 batch functionality.

Note about 8.5.5.7 issues with feature incompatibility … and workarounds

The 8.5.5.7 level of Liberty introduced some issues with feature incompatibility when `zosLocalAdapters-1.0` and `batch-1.0` were both present in the `<featureManager>` list in the `server.xml` file.

You can work around this issue in one of three ways:

1. Do not code `zosLocalAdapters-1.0` in `<featureManager>`

   This is needed only if you are using the batchManagerZos function (which we illustrate on page 39). If `zosLocalAdapters-1.0` is not in the `<featureManager>` list then the incompatible feature errors will not present themselves.

2. As a workaround, use `installUtility` to install the following two *additional* features:

   ```
   installUtility install ejbLite-3.2
   installUtility install jca-1.7
   ```

   That overcomes the incompatibility issue and will allow `batch-1.0` and `zosLocalAdapters-1.0` to coexist in the `<featureManager>` list.

3. Rather than install the features individually as illustrated earlier, use the zosBundle option to install everything at once:

   ```
   installUtility install zosBundle
   ```

   This will install a long list of features, including all you need for JSR 352 as well as the two workaround features shown under #2. The ability to install a "bundle" provides a way to easily install many features at once.

SQLCODE = -633, ERROR: THE DELETE RULE MUST BE CASCADE OR NO ACTION

If when implementing the DDL for jobRepository persistence with DB2 z/OS you get the following message:

```
DSNT408I SQLCODE = -633, ERROR:  THE DELETE RULE MUST BE CASCADE OR NO ACTION
DSNT418I SQLSTATE   = 42915 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXICKD SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = 30 0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'0000001E'  X'00000000'  X'00000000'  X'FFFFFFFF'
         X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION
```

That is most likely due to your DB2 system's CURRENT RULES setting and the following ALTER TABLE statement in the generated DDL:

```
ALTER TABLE JBATCH.STEPTHREADEXECUTION ADD CONSTRAINT
  STPTHFKTPLVLSTPXCD FOREIGN KEY (FK_TOPLVL_STEPEXECID)
  REFERENCES JBATCH.STEPTHREADEXECUTION (STEPEXECID);
```

That references itself. To correct that, you can add the following:

```
ALTER TABLE JBATCH.STEPTHREADEXECUTION ADD CONSTRAINT
  STPTHFKTPLVLSTPXCD FOREIGN KEY (FK_TOPLVL_STEPEXECID)
  REFERENCES JBATCH.STEPTHREADEXECUTION (STEPEXECID)
  ON DELETE NO ACTION;
```

Then re-run the SQL.

If your z/OS LPAR does not have internet access

This involves downloading the fixes to some system or workstation that has internet access, then getting the ZIP file up to your z/OS LPAR to create a local repository. Then the `installUtility` function can access the features and install them locally.

Do the following:

- ☐ Go the IBM Fix Central[59] and locate the Liberty feature repository. For 8.5.5.6 it has a name of `wlp-featureRepo-8.5.5.6.zip`. It is about 200MB in size[60].

59 URL: `https://www.ibm.com/support/fixcentral/`

60 This contains all the features. That's more than you need for batch, but you may want other features later. With this you will have them.

- ☐ Upload that ZIP file to you z/OS LPAR and unzip. Note the directory path as you will point to this from the `installUtility` function.
- ☐ Create a directory called `/etc` under the Liberty Profile install directory. This will be peer to the `/bin` directory in which the `installUtility` function resides.
- ☐ On your workstation, create a text file called `repositories.txt` and populate with the following:

  ```
  loc-rep.url=file:///<path>/
  ```

  Where the *<path>* is to the directory in UNIX systems services into which you unzipped the features ZIP file.
- ☐ Save the file.
- ☐ Upload that file in binary to your z/OS system and place in the `/etc` directory you created a few steps ago.
- ☐ Rename the file to `repositories.properties`
- ☐ Recursively change the file ownership to your ID and group Installation Manager used when it installed Liberty, for example:

  ```
  chown -R IMADMIN:IMGROUP /<path>/etc
  ```
- ☐ Recursively change the file permissions to 775, for example:

  ```
  chmod – R 775 /<path>/etc
  ```
- ☐ Open a Telnet (or OMVS) session and login with your IM admin ID. Go to the `/bin` directory of your Liberty install.
- ☐ Issue the following command to see the settings for the `installUtility` function:

  `./installUtility viewSettings`

  You should see something like this:

  ```
  installUtility Settings
  ---------------------------------------------------------------------
  Properties File:
  /<path>/etc/repositories.properties
  Default Assets Repository: IBM WebSphere Liberty Repository
  Use Default Repository: True

  Configured Repositories
  ---------------------------------------------------------------------
  Name: loc-rep
  Location: file:///<path>/
  User Name: <Not Applicable>
  Password: <Not Applicable>

  Proxy Settings
  ---------------------------------------------------------------------
  No proxy configured
  ```
- ☐ Verify the "Properties File" it points to is the one you created and placed in the /etc directory.
- ☐ Verify the "Location" under "Configured Repositories" points to your path to where you unzipped the ZIP file from Fix Central.
- ☐ If the "viewSettings" command verified your locations, then issue the following command to install the `batch-1.0` feature:

  `./installUtility install batch-1.0 --acceptLicense`

  You should see something like this:

```
Establishing a connection to the configured repositories...
This process might take several minutes to complete.

Successfully connected to all configured repositories.

Resolving remote assets. This process might take several minutes to
complete.
The --acceptLicense argument was found. This indicates that you have
accepted the terms of the license agreement.


Step 1 of 7: Downloading jdbc-4.1...
Step 2 of 7: Downloading servlet-3.1...
Step 3 of 7: Downloading batch-1.0...
Step 4 of 7: Installing jdbc-4.1...
Step 5 of 7: Installing servlet-3.1...
Step 6 of 7: Installing batch-1.0...
Step 7 of 7: Cleaning up temporary files...

One or more features installed successfully: jdbc-4.1 servlet-3.1 batch-1.0.
Start product validation...
Product validation completed successfully.
```

- ☐ Then issue the following command to install `batchManagement-1.0`:

  `./installUtility install batchManagement-1.0 --acceptLicense`

  You should see the following:

```
Step 1 of 3: Downloading batchManagement-1.0...
Step 2 of 3: Installing batchManagement-1.0...
Step 3 of 3: Cleaning up temporary files...

One or more features installed successfully: batchManagement-1.0.
```

- ☐ Then issue the following command to install `zosLocalAdapters-1.0`:

  `./installUtility install zosLocalAdapters-1.0 --acceptLicense`

  You should see the following:

```
Step 1 of 5: Downloading jca-1.6...
Step 2 of 5: Downloading zosLocalAdapters-1.0...
Step 3 of 5: Installing jca-1.6...
Step 4 of 5: Installing zosLocalAdapters-1.0...
Step 5 of 5: Cleaning up temporary files...

One or more features installed successfully: jca-1.6 zosLocalAdapters-1.0.
```

- ☐ You have installed the necessary features.

Sample XML

Batch persistence – Derby

The following XML assumes the Derby database is created with the name `BATCHDB` under the server's `/resources` directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
        <feature>appSecurity-2.0</feature>
    </featureManager>
    <keyStore id="defaultKeyStore" password="Liberty"/>

    <basicRegistry id="basic1" realm="jbatch">
       <user name="Fred" password="fredpwd" />
    </basicRegistry>

    <authorization-roles id="com.ibm.ws.batch">
       <security-role name="batchAdmin">
          <user name="Fred" />
       </security-role>
    </authorization-roles>

    <batchPersistence jobStoreRef="BatchDatabaseStore" />

    <databaseStore id="BatchDatabaseStore"
      dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />

    <library id="DerbyLib">
       <fileset dir="${server.config.dir}/resources/derby" />
    </library>

    <dataSource id="batchDB"
       jndiName="jdbc/batch"
       type="javax.sql.XADataSource">

     <jdbcDriver libraryRef="DerbyLib" />

     <properties.derby.embedded
       databaseName="${server.config.dir}/resources/BATCHDB"
       createDatabase="create"
       user="user"
       password="pass" />
    </dataSource>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="9080"
                  httpsPort="9443" />

</server>
```

Batch persistence – DB2 z/OS

The following XML assumes the use of a JDBC Type 4 driver[61]. Those areas needing customization are highlighted in *<blue>*.

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
        <feature>appSecurity-2.0</feature>
    </featureManager>

    <keyStore id="defaultKeyStore" password="Liberty"/>
```

61 Type 2 *can* be used. Type 4 is easier to start with because it does not require the Liberty Profile z/OS Angel process.

```
    <basicRegistry id="basic1" realm="jbatch">
       <user name="Fred" password="fredpwd" />
    </basicRegistry>

    <authorization-roles id="com.ibm.ws.batch">
       <security-role name="batchAdmin">
          <user name="Fred" />
       </security-role>
    </authorization-roles>

    <batchPersistence jobStoreRef="BatchDatabaseStore" />

    <databaseStore id="BatchDatabaseStore"
      dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />

    <jdbcDriver id="DB2T4" libraryRef="DB2T4LibRef" />

    <library id="DB2T4LibRef">
      <fileset dir="/<path>/jdbc/classes/"
      includes="db2jcc4.jar db2jcc_license_cisuz.jar sqlj4.zip" />
    </library>

    <authData id="batchAlias" user="<userid>" password="<password>" />

    <dataSource id="batchDB"
       containerAuthDataRef="batchAlias"
       type="javax.sql.XADataSource"
       jdbcDriverRef="DB2T4">
      <properties.db2.jcc
       serverName="<host>"
       portNumber="<port>"
       databaseName="<location_name>"
       driverType="4" />
    </dataSource>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="9080"
                  httpsPort="9443" />

</server>
```

Batch persistence and BonusPayout Sample – Derby/Derby

This sample shows the JSR persistence using Derby and the Bonus application using Derby as well. One library reference, but two datasources:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
        <feature>appSecurity-2.0</feature>
    </featureManager>

    <keyStore id="defaultKeyStore" password="Liberty"/>
    <basicRegistry id="basic1" realm="jbatch">
       <user name="Fred" password="fredpwd" />
    </basicRegistry>

    <authorization-roles id="com.ibm.ws.batch">
       <security-role name="batchAdmin">
          <user name="Fred" />
       </security-role>
    </authorization-roles>

    <batchPersistence jobStoreRef="BatchDatabaseStore" />

    <databaseStore id="BatchDatabaseStore"
      dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />

    <library id="DerbyLib">
       <fileset dir="${server.config.dir}/resources/derby" />
    </library>

    <dataSource id="batchDB"
       jndiName="jdbc/batch"
       type="javax.sql.XADataSource">

     <jdbcDriver libraryRef="DerbyLib" />

     <properties.derby.embedded
        databaseName="${server.config.dir}/resources/BATCHDB"
        createDatabase="create"
        user="user"
        password="pass" />
    </dataSource>

    <dataSource id="bonusDB"
       jndiName="jdbc/bonus"
       type="javax.sql.XADataSource">

      <jdbcDriver libraryRef="DerbyLib" />

      <properties.derby.embedded
         databaseName="${server.config.dir}/resources/BONUSDB"
         createDatabase="create"
         user="user" password="pass" />
    </dataSource>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="9080"
                  httpsPort="9443" />

</server>
```

Batch persistence and BonusPayout Sample – Derby/DB2

This sample shows the JSR persistence using Derby and the Bonus application using DB2. This implies two sets of library and datasource definitions:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
        <feature>appSecurity-2.0</feature>
    </featureManager>

    <keyStore id="defaultKeyStore" password="Liberty"/>

    <basicRegistry id="basic1" realm="jbatch">
       <user name="Fred" password="fredpwd" />
    </basicRegistry>

    <authorization-roles id="com.ibm.ws.batch">
       <security-role name="batchAdmin">
          <user name="Fred" />
       </security-role>
    </authorization-roles>

    <batchPersistence jobStoreRef="BatchDatabaseStore" />

    <databaseStore id="BatchDatabaseStore"
      dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />

    <library id="DerbyLib">
       <fileset dir="${server.config.dir}/resources/derby" />
    </library>

    <dataSource id="batchDB"
       jndiName="jdbc/batch"
       type="javax.sql.XADataSource">

     <jdbcDriver libraryRef="DerbyLib" />

     <properties.derby.embedded
       databaseName="${server.config.dir}/resources/BATCHDB"
       createDatabase="create"
       user="user"
       password="pass" />
    </dataSource>

    <jdbcDriver id="DB2T4" libraryRef="DB2T4LibRef" />

    <library id="DB2T4LibRef">
      <fileset dir="/<path>/jdbc/classes/"
      includes="db2jcc4.jar db2jcc_license_cisuz.jar sqlj4.zip" />
    </library>

    <authData id="bonusAlias" user="<userid>" password="<password>" />

    <dataSource id="bonusDB" jndiName="jdbc/bonus"
        containerAuthDataRef="bonusAlias"
        type="javax.sql.XADataSource"
        jdbcDriverRef="DB2T4">
      <properties.db2.jcc
        serverName="<host>"
        portNumber="<port>"
        databaseName="<location_name>"
        driverType="4" />
    </dataSource>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="9080"
                  httpsPort="9443" />

</server>
```

Batch persistence and BonusPayout Sample – DB2/Derby

This sample shows the JSR persistence using DB2 and the Bonus application using Derby. This implies two sets of library and datasource definitions:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
        <feature>appSecurity-2.0</feature>
    </featureManager>

    <keyStore id="defaultKeyStore" password="Liberty"/>

    <basicRegistry id="basic1" realm="jbatch">
       <user name="Fred" password="fredpwd" />
    </basicRegistry>

    <authorization-roles id="com.ibm.ws.batch">
       <security-role name="batchAdmin">
          <user name="Fred" />
       </security-role>
    </authorization-roles>

    <batchPersistence jobStoreRef="BatchDatabaseStore" />

    <databaseStore id="BatchDatabaseStore"
      dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />

    <jdbcDriver id="DB2T4" libraryRef="DB2T4LibRef" />

    <library id="DB2T4LibRef">
      <fileset dir="/<path>/jdbc/classes/"
      includes="db2jcc4.jar db2jcc_license_cisuz.jar sqlj4.zip" />
    </library>

    <authData id="batchAlias" user="<userid>" password="<password>" />
    <dataSource id="batchDB"
        containerAuthDataRef="batchAlias"
        type="javax.sql.XADataSource"
        jdbcDriverRef="DB2T4">
      <properties.db2.jcc
        serverName="<host>"
        portNumber="<port>"
        databaseName="<location_name>"
        driverType="4" />
    </dataSource>
    <library id="DerbyLib">
       <fileset dir="${server.config.dir}/resources/derby" />
    </library>

    <dataSource id="bonusDB"
        jndiName="jdbc/bonus"
        type="javax.sql.XADataSource">
      <jdbcDriver libraryRef="DerbyLib" />
      <properties.derby.embedded
        databaseName="${server.config.dir}/resources/BONUSDB"
        createDatabase="create"
        user="user" password="pass" />
    </dataSource>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="9080"
                  httpsPort="9443" />

</server>
```

Batch persistence and BonusPayout Sample – DB2/DB2

This sample shows both the JSR persistence and the Bonus application using DB2 z/OS and JDBC Type 4. One library and two data sources:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
        <feature>appSecurity-2.0</feature>
    </featureManager>

    <keyStore id="defaultKeyStore" password="Liberty"/>

    <basicRegistry id="basic1" realm="jbatch">
       <user name="Fred" password="fredpwd" />
    </basicRegistry>

    <authorization-roles id="com.ibm.ws.batch">
       <security-role name="batchAdmin">
          <user name="Fred" />
       </security-role>
    </authorization-roles>

    <batchPersistence jobStoreRef="BatchDatabaseStore" />

    <databaseStore id="BatchDatabaseStore"
      dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />

    <jdbcDriver id="DB2T4" libraryRef="DB2T4LibRef" />

    <library id="DB2T4LibRef">
      <fileset dir="/<path>/jdbc/classes/"
      includes="db2jcc4.jar db2jcc_license_cisuz.jar sqlj4.zip" />
    </library>

    <authData id="batchAlias" user="<userid>" password="<password>" />

    <dataSource id="batchDB"
       containerAuthDataRef="batchAlias"
       type="javax.sql.XADataSource"
       jdbcDriverRef="DB2T4">
      <properties.db2.jcc
       serverName="<host>"
       portNumber="<port>"
       databaseName="<location_name>"
       driverType="4" />
    </dataSource>

    <authData id="bonusAlias" user="<userid>" password="<password>" />

    <dataSource id="bonusDB" jndiName="jdbc/bonus"
       containerAuthDataRef="bonusAlias"
       type="javax.sql.XADataSource"
       jdbcDriverRef="DB2T4">
      <properties.db2.jcc
       serverName="<host>"
       portNumber="<port>"
       databaseName="<location_name>"
       driverType="4" />
    </dataSource>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="9080"
                  httpsPort="9443" />

</server>
```

A note about undeliverable messages

Queueing solutions, such as the Liberty "default messaging provider" (sometimes called the SIBus), or IBM MQ, have a mechanism to define where undeliverable messages should go. This is important because if a message is undeliverable, then *something* must be done with it, otherwise it will remain in the queue and potentially cause problems. In this section we will briefly describe how to define what to do with undeliverable messages.

Liberty default messaging provider

We used this earlier in the document (starting on page 42) for the queue manager for the multi-JVM design. The Knowledge Center article to reference for configuring undeliverable messages is:

https://www.ibm.com/support/knowledgecenter/en/SSAW57_liberty/com.ibm.websphere.liberty.autogen.nd.doc/ae/rwlp_config_messagingEngine.html

The queue element allows for three definitions related to undeliverable messages:

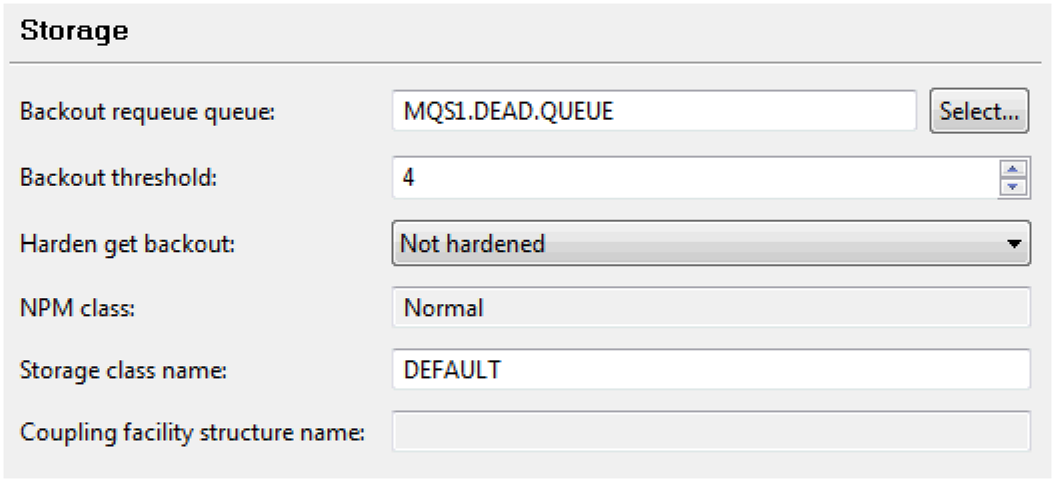
failedDeliveryPolicy	This defines what to do with a message that can't be delivered. The default is to SEND_TO_EXCEPTION_DESTINATION, which is what we would recommend.
maxRedeliveryCount	This defines how many delivery attempts should be tried before invoking the failedDeliveryPolicy. The default is 5.

exceptionDestination	This defins where failed messages are to go if the failedDeliveryPolicy indicates failed messages are to be sent to a queue. The default is _SYSTEM.Exception.Destination, but you can change it to a defined queue of your choice if you wish.

By default a queue defined to the internal messaging function of Liberty will send undeliverable messages to a defined exception queue after five delivery attempts.

IBM MQ

For MQ, the common practice is to define a "dead letter queue," and to define on the queue between the Executor and Dispatcher the name of this dead letter queue. In MQ Explorer, that would look like this:



Storage

Backout requeue queue: MQS1.DEAD.QUEUE

Backout threshold: 4

Harden get backout: Not hardened

NPM class: Normal

Storage class name: DEFAULT

Coupling facility structure name:

The "Backout requeue queue" defines the destination for undeliverable messages, and the "backout threshold" defines how many delivery attempts should be made before giving up and sending the message to the backout queue. If you leave these fields undefined, then undeliverable messages will remain in the queue until cleared manually. This can creation operational problems for the JSR-352 batch environment. That is why it is a good practice to define a "dead letter queue" so MQ can handle undeliverable messages.

Using a REST client to submit and monitor jobs

In this section we will illustrate the use of a browser-based REST client to submit and monitor jobs. As stated earlier, we anticipate most will use the batchManager command line client for this. But if you're interested in seeing a REST client in action, follow these steps.

- ☐ Install the REST client into your Firefox browser. See “The Firefox REST client” on page 81 for more on installing that.
- ☐ Close all instances of your Firefox browser and re-open. We do this so our initial connections to the Liberty Profile server do not try to use previous HTTP connections and cookies.
- ☐ From a *standard* Firefox tab[62] (*not* the REST client), issue the following URL:

  ```
  https://<host>:<secure_port>/ibm/api/batch/jobinstances/
  ```

  Where:

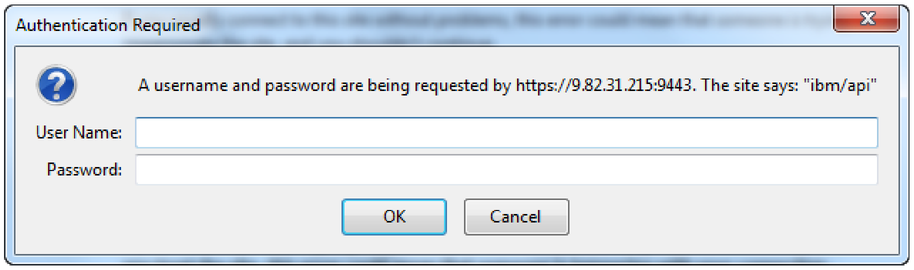
  - The protocol is `https`, not merely `http`
  - The *<host>* value is your server TCP host name

62 The REST client does not handle establishment of SSL connections where the CA certificate is a self-signed certificate, which is what Liberty Profile is going to use in this case. We do this so you can manually accept the security challenge. Then the REST client will work.

- The **`<secure_port>`** value is the “https” port for your server (which defaults to 9443 if that is the value you are using)

☐ Accept the security challenge[63].

☐ You will be prompted for an ID and password:


Authentication Required

A username and password are being requested by https://9.82.31.215:9443. The site says: "ibm/api"

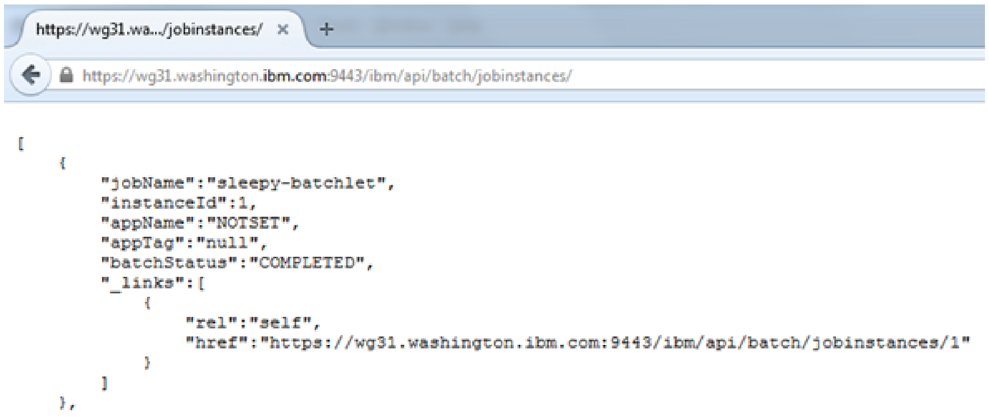
User Name:

Password:

This is because the IBM JSR 352 REST interface implementation is marked protected and requires authentication.

☐ Supply the ID and password you supplied in the `<basicRegistry>` element in your `server.xml` file – `Fred` and `fredpwd`. Note: that is *case sensitive*.

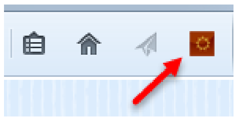
☐ You should get back a listing of previous job instances. It will look something like this:


https://wg31.wa.../jobinstances/
https://wg31.washington.ibm.com:9443/ibm/api/batch/jobinstances/

```
[
    {
        "jobName":"sleepy-batchlet",
        "instanceId":1,
        "appName":"NOTSET",
        "appTag":"null",
        "batchStatus":"COMPLETED",
        "_links":[
            {
                "rel":"self",
                "href":"https://wg31.washington.ibm.com:9443/ibm/api/batch/jobinstances/1"
            }
        ]
    },
```

Note: You will likely have more than one job instance in the list. The command you entered queries the job repository data store and returns all the job instances retrieved.

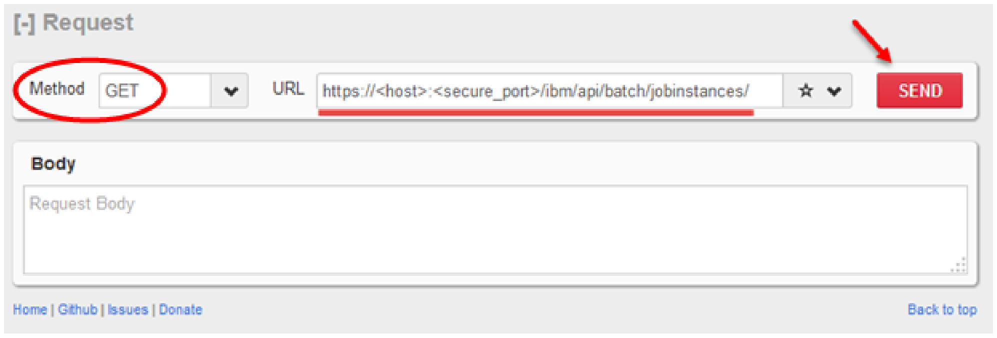
☐ Click on the REST client icon:



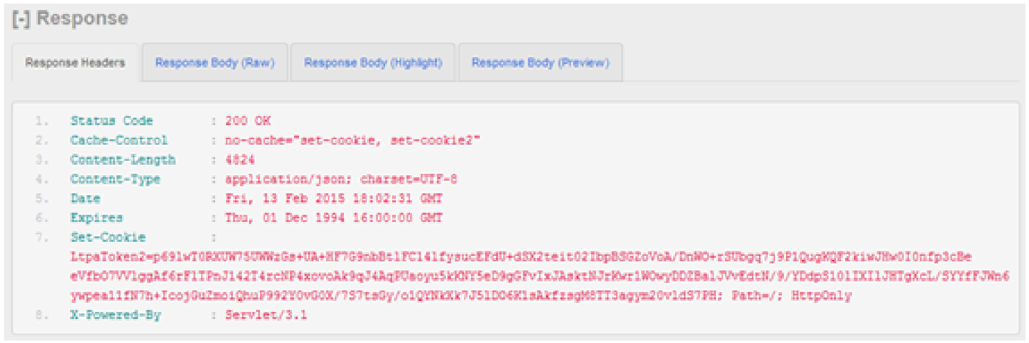
That will open a new tab with the REST client interface.

☐ Make sure the “Method” is GET, put the same URL before and click the SEND button:

63 Make sure the host and port is *your* Liberty Profile server. Accepting security challenges like this should only be done when you are certain the host can be trusted.

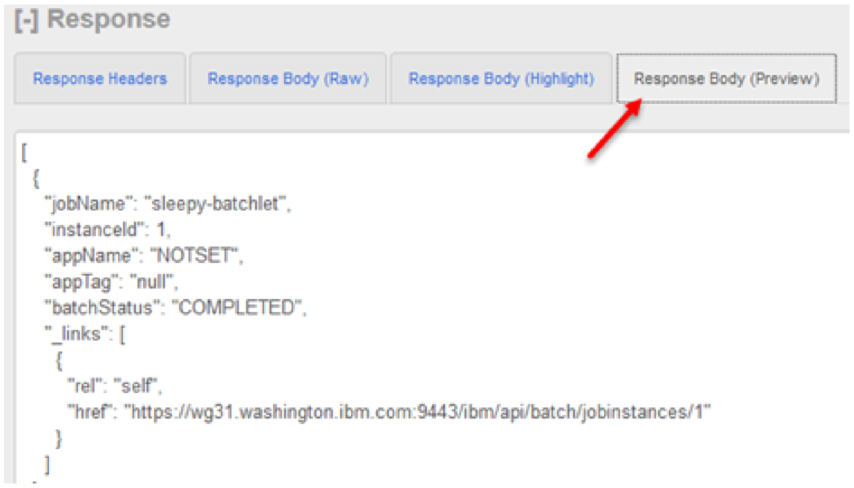

[-] Request
Method
GET
URL
https://<host>:<secure_port>/ibm/api/batch/jobinstances/
SEND
Body
Request Body
Home | Github | Issues | Donate
Back to top

- ☐ You will again be prompted for a userid and password. Supply the same value as before: `Fred` and `fredpwd`. That is case sensitive.
- ☐ You should receive a Status Code of "200 OK" and something that looks like this:



```
Status Code       : 200 OK
Cache-Control     : no-cache="set-cookie, set-cookie2"
Content-Length    : 4824
Content-Type      : application/json; charset=UTF-8
Date              : Fri, 13 Feb 2015 18:02:31 GMT
Expires           : Thu, 01 Dec 1994 16:00:00 GMT
Set-Cookie        :
LtpaToken2=p691wT0RXUW75UWWzGs+UA+HF7G9nbBt1FC141fysucEFdU+dSX2teit02IbpBSGZoVoA/DnWO+rSUbgq7j9P1QugKQF2kiwJHw0I0nfp3cBe
eVfbO7VVlggAf6rF1TPnJ142T4rcNP4xovoAk9qJ4AqPUaoyu5kKNY5eD9gGFvIxJAsktNJrKwr1WOwyDDZBalJVvEdtN/9/YDdpS101IXI1JHTgXcL/SYYfFJWn6
ywpea11fN7h+IcojGuZmoiQhuP992Y0vG0X/7S7tsGy/o1QYNkXk7J51DO6K1sAkfzsgM8TT3agym20v1dS7PH: Path=/; HttpOnly
X-Powered-By      : Servlet/3.1
```

- ☐ If you click on the "Response Body (Preview)" tab you will see the JSON listing of the job instances, just as you did before:



```
[
 {
  "jobName": "sleepy-batchlet",
  "instanceId": 1,
  "appName": "NOTSET",
  "appTag": "null",
  "batchStatus": "COMPLETED",
  "_links": [
   {
    "rel": "self",
    "href": "https://wg31.washington.ibm.com:9443/ibm/api/batch/jobinstances/1"
   }
  ]
```

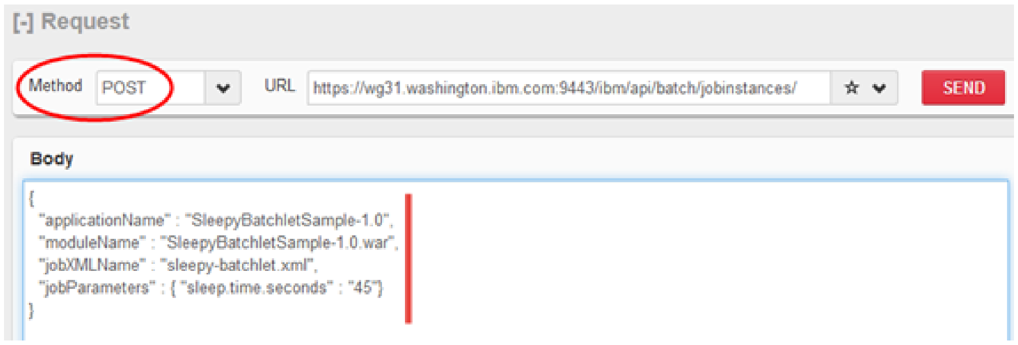
That was setup to get to what we really want to do, which is to submit a job through this REST client. That's next.

- ☐ Set the "Method" to POST, keep the URL the same as before, and put the following JSON[64] into the "Body" field[65], then click the SEND button:

```
{
  "applicationName" : "SleepyBatchletSample-1.0",
  "moduleName" : "SleepyBatchletSample-1.0.war",
  "jobXMLName" : "sleepy-batchlet.xml",
```

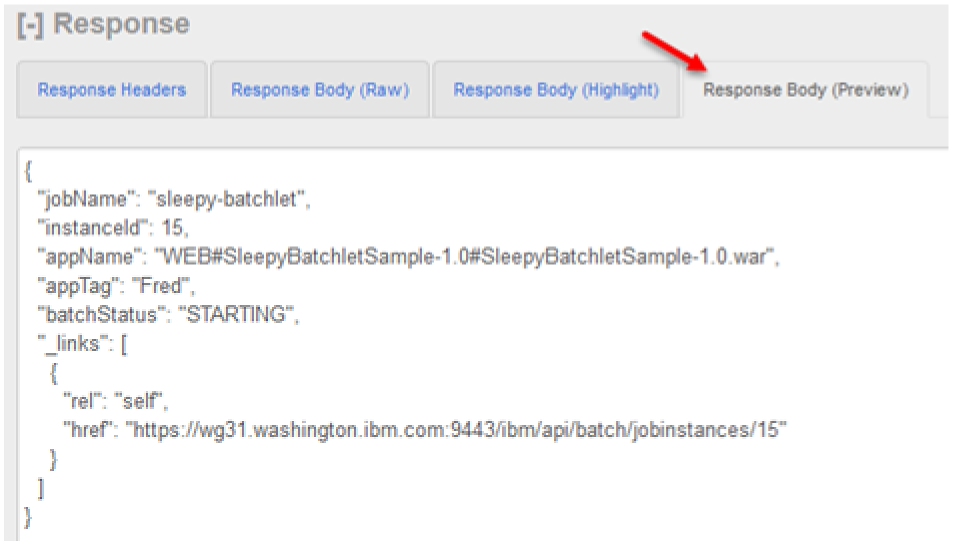
64 See "The JSON format for job submission through REST interface" on page 81 for more on this JSON object.
65 Copy/paste from the PDF may result in unacceptable characters. If you see a failure, compose the JSON in Notepad and copy/paste from there. In particular, be sure the brace and double quote symbols are standard left and right brace, and straight double-quote and not the "smart quotes" where they angled or where the first double quote is right-side-up and the trailing is upside-down.

```
  "jobParameters" : { "sleep.time.seconds" : "45"}
}
```


[-] Request
Method
POST
URL
https://wg31.washington.ibm.com:9443/ibm/api/batch/jobinstances/
SEND
Body

```
{
 "applicationName" : "SleepyBatchletSample-1.0",
 "moduleName" : "SleepyBatchletSample-1.0.war",
 "jobXMLName" : "sleepy-batchlet.xml",
 "jobParameters" : { "sleep.time.seconds" : "45"}
}
```

☐ You should see the following:



```
{
 "jobName": "sleepy-batchlet",
 "instanceId": 15,
 "appName": "WEB#SleepyBatchletSample-1.0#SleepyBatchletSample-1.0.war",
 "appTag": "Fred",
 "batchStatus": "STARTING",
 "_links": [
   {
     "rel": "self",
     "href": "https://wg31.washington.ibm.com:9443/ibm/api/batch/jobinstances/15"
   }
 ]
}
```

That is confirmation the job was submitted and is starting, just as it did before when you submitted it through the servlet.

☐ Take a look at the JSON that was returned and note the "instanceID" value. Use the REST client again and send the following:

Method:	`GET`
URL:	`https://<host>:<secure_port>/ibm/api/batch/jobexecutions/#` Where `<host>` and `<secure_port>` are the same as before, and `#` is the instanceID value from your job.

You will get back information about the job execution.

☐ Now append `/joblogs` to the end of the URL you used in the previous step:

`https://<host>:<secure_port>/ibm/api/batch/jobexecutions/#/joblogs`

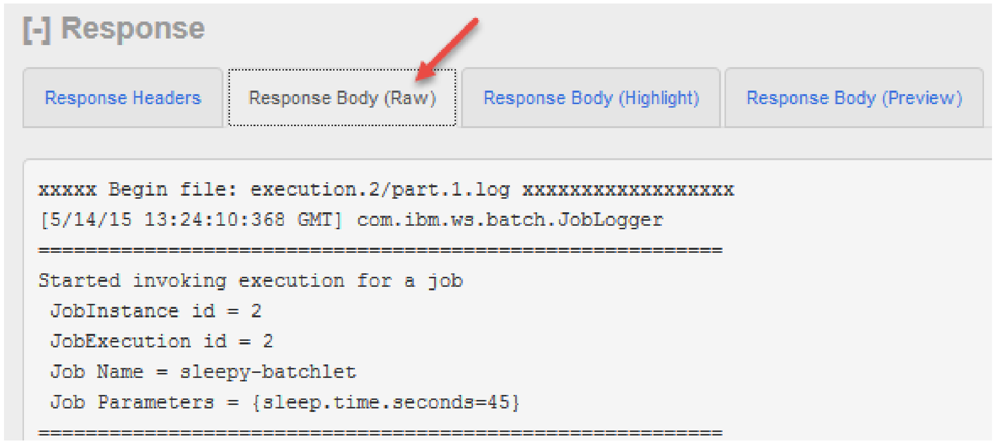
and click the SEND button. That will return a list of URLs that may be used to retrieve the job log – either as text or as a zip-format file, and all the parts combined or individual job part parts:

```
[
  {
    "rel": "joblog text",
    "href": "https://<host>:<port>/ibm/api/batch/jobexecutions/15/joblogs?type=text"
  },
  {
    "rel": "joblog zip",
    "href": "https://<host>:<port>/ibm/api/batch/jobexecutions/15/joblogs?type=zip"
  },
  {
    "rel": "joblog part text",
    "href": "https://<host>:<port>/ibm/api/batch/jobexecutions/15/joblogs?
part=part.1.log&type=text"
  },
  {
    "rel": "joblog part zip",
    "href": "https://<host>:<port>/ibm/api/batch/jobexecutions/15/joblogs?
part=part.1.log&type=zip"
  }
]
```

☐ Append `?type=text` on the end of the URL you used in the previous step.

`https://<host>:<secure_port>/ibm/api/batch/jobexecutions/#/joblogs?type=text`

and click the SEND button. That will return the job log in text format to the "Response Body (raw)" tab of the REST client:



```
xxxxx Begin file: execution.2/part.1.log xxxxxxxxxxxxxxxxxx
[5/14/15 13:24:10:368 GMT] com.ibm.ws.batch.JobLogger
==========================================================
Started invoking execution for a job
 JobInstance id = 2
 JobExecution id = 2
 Job Name = sleepy-batchlet
 Job Parameters = {sleep.time.seconds=45}
==========================================================
```

At this point you see how the REST interface works – it exposes a set of REST URI patterns that you may use to submit, monitor and control a job. The details of the APIs are provided at the Knowledge Center article found by searching on the unique string `rwlp_batch_rest_api`.

Some quick notes to finish this section:

- In the "real world" you would not use a browser plugin REST client. We use that simply because it's easy to get and use. In actual practice a programmatic REST will be used access the IBM JSR 352 REST client.
- There are a few more REST API patterns documented at the Knowledge Center article:

  `PUT .../ibm/api/batch/jobinstances/#?action=stop`

  This will stop a running job based on the job instance ID (**#**) you supply.

  `PUT .../ibm/api/batch/jobinstances/#?action=restart`

  This will restart a stopped job based on the job instance ID (**#**) you supply.

  `DELETE .../ibm/api/batch/jobinstances/#`

  This will delete from the job repository data store information about the job instance, as well as all job logs associated with the job instance

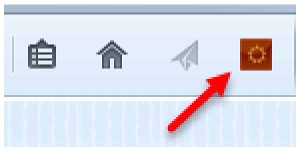
The Firefox REST client

To use the REST interface of the IBM JSR 352 implementation requires a REST client. Many REST clients exist for your use. The one we will use for this document is the Firefox REST client[66], which can be found here:

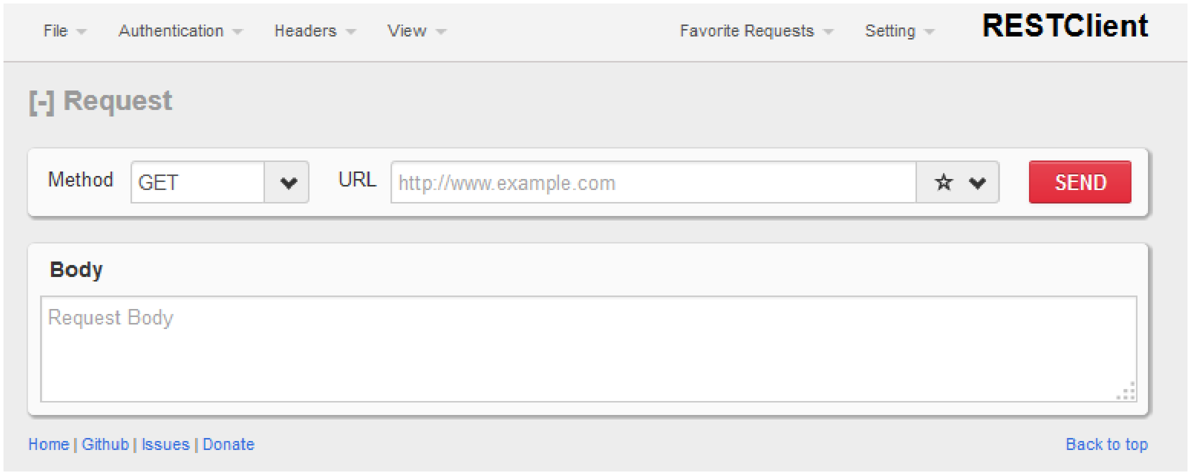
https://addons.mozilla.org/en-US/firefox/addon/restclient/

Go that website and click the "Add to Firefox" button:

When it is added to your Firefox installation, you will see a little icon like this:



And when you click on that icon you will see the following:



That is the interface we will use in this document when we refer to a "browser REST client."

The JSON format for job submission through REST interface

Earlier we showed the following JSON object for submitting the SleepyBatchlet application:

```
{
  "applicationName" : "SleepyBatchletSample-1.0",
  "moduleName" : "SleepyBatchletSample-1.0.war",
  "jobXMLName" : "sleepy-batchlet.xml",
  "jobParameters" : { "sleep.time.seconds" : "45"}
}
```

That format comes from the Knowledge Center article that can be found by searching on the unique string **`rwlp_batch_rest_api`**. That article details the REST API support for the IBM JSR 352 implementation.

The JSON elements are as follows:

66 This is for illustration purposes only. You are free to use whatever REST client you wish. This does not imply an endorsement of Mozilla, Firefox or this particular REST client.

Element	*Required?*	*Notes*
`applicationName`	No[67]	This identifies the Java batch application to be invoked. If this is omitted, then the `moduleName` must be supplied, and the application name becomes the module name with `.war` stripped off.
`moduleName`	No[68]	This identifies the WAR file in which the Java batch application is packaged. If this is omitted, the `applicationName` value is appended with `.war`.
`jobXMLName`	Yes	This is required. This indicates the job specification language XML file that will be used to submit this job. If you look in the `SleepyBatchletSample-1.0.war` file, you'll see this XML there with name `sleepy-batchlet`.xml.
`jobParameters`	No	This is used to pass in parameters to the job. The SleepyBatchlet sample has one parameter – `sleep.time.seconds`. You can string multiple paramters by separating each name/value pair with a comma.

Therefore, the JSON we had you use was based on the pattern shown in the Knowledge Center article, with the specifics of the SleepyBatchlet sample provided.

The JSON for the BonusPayout would be this:

```
{
  "applicationName" : "BonusPayout-1.0",
  "moduleName" : "BonusPayout-1.0.war",
  "jobXMLName" : "SimpleBonusPayoutJob.xml",
  "jobParameters" : { "numRecords"   : "100" ,
                      "chunkSize"    : "10"  ,
                      "dsJNDI"       : "jdbc/bonus"   ,
                      "bonusAmount" : "123"   ,
                      "generateFileNameRoot" : "/<path>/<prefix>",
                      "tableName"    : "BONUSDB.ACCOUNT"
                    }
}
```

67 Either `applicationName` or `moduleName` must be entered. You may code both. But at least one must be entered.
68 Either `applicationName` or `moduleName` must be entered. You may code both. But at least one must be entered.

Reference Links

The following provides a list of useful URLs for more information on IBM JSR 352 Java Batch:

JSR 352 Specification Page

https://jcp.org/en/jsr/detail?id=352

IBM Knowledge Center

The IBM Knowledge Center contains a great deal of very good information about IBM products, including the IBM JSR 352 Java batch implementation. The Knowledge Center is organized as *articles* related to a specific topic.

The URL for the Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5

The table below provides a set of unique search strings that can be used to take you directly to the article for the topic cited.

Topic	***Unique Search String***
Deploying Java batch applications for the Liberty profile	`twlp_container_batch`
Java batch and managed batch overview	`cwlp_batch_overview`
Configuring the Liberty profile for the batch REST API	`twlp_batch_configrest`
Securing the Liberty profile batch environment	`twlp_batch_securing`
Java batch shutdown and recovery	`rwlp_batch_shutdown`
Java batch persistence configuration	`rwlp_batch_persistence_config`
Enabling multiple server support by using the Liberty profile embedded messaging provider	`twlp_batch_multiJVMembed`
Enabling multiple server support by using the WebSphere MQ messaging provider	`twlp_batch_multiJVMmq`
REST API administration	`rwlp_batch_rest_api`
batchManager command-line client utility	`cwlp_jbatch_commandlineutil`
Configuring the batchManagerZos client utility	`twlp_batchManagerZos`
Viewing Java batch job logs	`rwlp_batch_view_joblog`

GitHub

GitHub is the repository for the JSR 352 sample applications used in this document.

Here's a “getting started” link for items related to IBM JSR 352 Java Batch:

https://github.com/WASdev/sample.batch.templateddls/blob/master/GettingStarted.md

Here's a link for the SleepyBatchlet sample:

https://github.com/WASdev/sample.batch.sleepybatchlet

And here's a link for the BonusPayout sample:

https://github.com/WASdev/sample.batch.bonuspayout

“Doctor Batch Magic Sauce”

This is an application framework that allows Java batch applications to be portable between the IBM Compute Grid programming model and the newer JSR 352 programming model. Find information on this at GitHub:

https://github.com/WASdev/lib.batch.magicsauce

Document Change History

Check the date in the footer of the document for the version of the document.

July 8, 2015	Original document at time of Techdoc creation
October 19, 2015	Minor update on batchManager command to be more consistent with the Derby database name created a few pages earlier.
October 21, 2015	Provided information on 8.5.5.7 issues with feature incompatibility and workarounds to the issue. This is found in the "Miscellaneous" section.
November 10, 2016	Various minor updates.
January 25, 2017	Various updates based on review and feedback from Carl Farkas of IBM France. Including the misspelling of "Implementation" on the cover page! ☺
January 27,2017	Updated DB2 JDBC Type 4 examples to better show use of authentication alias.
November 14, 2018	Updated Multi-JVM section with information on defining the queue to process undeliverable messages. The bulk of the updates were provided in the miscellaneous section. See "A note about undeliverable messages" on page 75.

End of WP102544