

Administering batch environments, Version 8.5

Administering batch environments



Note

Before using this information, be sure to read the general information under “Notices” on page 261.

Compilation date: June 1, 2012

© Copyright IBM Corporation 2012.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

How to send your comments	vii
Using this PDF	ix
Chapter 1. Batch applications.	1
Batch overview	1
Learn about batch applications	1
Getting started with the batch environment	3
Chapter 2. Administering the batch environment	9
Configuring the batch environment	9
Environment planning for transactional batch applications and compute-intensive applications	9
Configuring the unit test environment (UTE) in Rational Application Developer	11
Configuring the job scheduler	11
Job scheduler WebSphere variables	12
Job scheduler System Programming Interfaces (SPI)	13
Creating the job scheduler and grid endpoint database.	15
Verifying the job scheduler installation	16
Securing the job scheduler	16
Job scheduler security overview	17
Securing the job scheduler using roles.	18
Securing the job scheduler using groups on distributed operating systems	20
Securing the job scheduler using roles and groups on distributed operating systems	21
Securing the job scheduler using groups on the z/OS operating system	22
Securing the job scheduler using roles and groups on the z/OS operating system	24
Configuring WebSphere grid endpoints	25
Endpoint WebSphere variables	26
Running batch jobs under user credentials	27
Batch jobs and their environment.	28
Job management console	29
Command-line interface for batch jobs	30
Job logs	42
Job classes	44
Creating and managing reports for batch statistics	45
Job scheduler integration with external schedulers	46
Integration of an external workload scheduler to manage batch workloads	46
Configuring the external scheduler interface.	47
Setting up the external scheduler interface using the default messaging provider	47
Setting up the external scheduler interface using WebSphere MQ	51
Requirements-based job scheduling.	53
Service policies for batch jobs	54
Batch job classification	55
Job usage data for charge-back accounting support.	57
Integrating batch features in z/OS operating systems	59
z/OS workload management and service policies.	60
Transaction class propagation on z/OS operating systems	60
Managing multi-user WLM environments	61
Managing worker threads	61
Enabling job usage information	62
Rolling out batch application editions	63
Job scheduler custom properties	64
MaxConcurrentDispatchers	64
UseHTTPSConnection	64

RECORD_SMF_SUBTYPES	65
JOB_SECURITY_POLICY	65
JOB_SECURITY_DEFAULT_GROUP	65
JOB_SECURITY_ADMIN_GROUP	66
UseAPCEndpointSelection	66
WXDBulletinBoardProviderOption	66
Port number settings for batch.	66
Batch administrator examples	68
xJCL sample for a batch job	68
XML schema for a batch job	70
xJCL sample for a compute intensive job	72
XML schema for a compute intensive job	73
xJCL sample for a native execution job	74
XML schema for a native execution job	74
CommandRunner utility job step	75
WSGrid properties file examples	78
Example: Jobs from repository properties file	78
Example: Compute-intensive properties file	78
Example: Transactional batch properties file.	78
Example: Restart job properties file	79
Example: xJCL file	80
Example: Control file	80
Chapter 3. Scripting batch applications	81
jobrecovery.batl.sh batch script	81
uteconfig.batl.sh batch script	81
configCGSharedLib.py batch script	82
removePGC.py batch script.	83
redeployLRS.py batch script	83
wsgridConfig.py batch script	84
JobSchedulerCommands command group for the AdminTask object	85
Chapter 4. Developing batch applications	91
Transactional batch and compute-intensive batch programming models	91
COBOL container overview	91
Developing COBOL container batch applications	93
Creating a COBOL call stub Java class	93
Compiling COBOL call stub Java classes.	94
Dynamically updating a COBOL module	94
COBOL call stub Java class usage example	94
COBOL RETURNING, RETURN-CODE, getReturnValue, and getReturnCode parameters.	95
COBOL container for batch troubleshooting	96
Generating COBOL call stubs	96
Creating a call stub generator configuration file	98
Invoking the call stub generator from a command line	101
Invoking the call stub generator from an Ant task	103
Invoking the call stub generator from a graphical interface	106
Call stub generator CSG.xml file	107
Call stub generator CSGBatch.xml file	110
Developing a simple compute-intensive application.	111
Compute-intensive programming model	113
Developing a simple transactional batch application	115
Components of a batch application	120
Batch programming model.	121
Skip-record processing	127
Retry-step processing	129

Configurable transaction mode	131
Developing a parallel job management application	131
Parallel job manager (PJM)	133
Parallel job manager application programming interfaces (APIs)	135
Other considerations for the parallel job manager	136
Using the batch data stream framework	136
Batch data stream framework and patterns	138
Implementing the generic batch step (GenericXDBatchStep)	157
Implementing the error tolerant step	158
Declaring the percentage-based threshold policy (PercentageBasedThresholdPolicy)	159
Declaring the record based threshold policy (RecordBasedThresholdPolicy)	160
Chapter 5. Deploying batch applications	161
Packaging EJB modules in a batch application using Rational Application Developer	161
Installing the batch application	161
Deploying an OSGi batch application	162
OSGi batch applications	164
Submitting batch jobs	164
xJCL elements	165
Submitting batch jobs using the job scheduler EJB interface	171
Submitting batch jobs using the job scheduler web service interface	192
Submitting jobs from an external job scheduler	248
Chapter 6. Troubleshooting batch applications	257
Adding log and trace settings to the batch environment	257
Batch common problems	257
Diagnosing batch problems using job logs	258
BusinessGridStatsCache log file	259
Notices	261
Trademarks and service marks	263
Index	265

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
 1. Display the article in your Web browser and scroll to the end of the article.
 2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an email form appears.
 3. Fill out the email form as instructed, and submit your feedback.
- To send comments on PDF books, you can email your comments to: **wasdoc@us.ibm.com**.

Your comment should pertain to specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer. When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about your comments.

Using this PDF

Links

Because the content within this PDF is designed for an online information center deliverable, you might experience broken links. You can expect the following link behavior within this PDF:

- Links to Web addresses beginning with `http://` work.
- Links that refer to specific page numbers within the same PDF book work.
- The remaining links will *not* work. You receive an error message when you click them.

Print sections directly from the information center navigation

PDF books are provided as a convenience format for easy printing, reading, and offline use. The information center is the official delivery format for IBM WebSphere Application Server documentation. If you use the PDF books primarily for convenient printing, it is now easier to print various parts of the information center as needed, quickly and directly from the information center navigation tree.

To print a section of the information center navigation:

1. Hover your cursor over an entry in the information center navigation until the **Open Quick Menu** icon is displayed beside the entry.
2. Right-click the icon to display a menu for printing or searching your selected section of the navigation tree.
3. If you select **Print this topic and subtopics** from the menu, the selected section is launched in a separate browser window as one HTML file. The HTML file includes each of the topics in the section, with a table of contents at the top.
4. Print the HTML file.

For performance reasons, the number of topics you can print at one time is limited. You are notified if your selection contains too many topics. If the current limit is too restrictive, use the feedback link to suggest a preferable limit. The feedback link is available at the end of most information center pages.

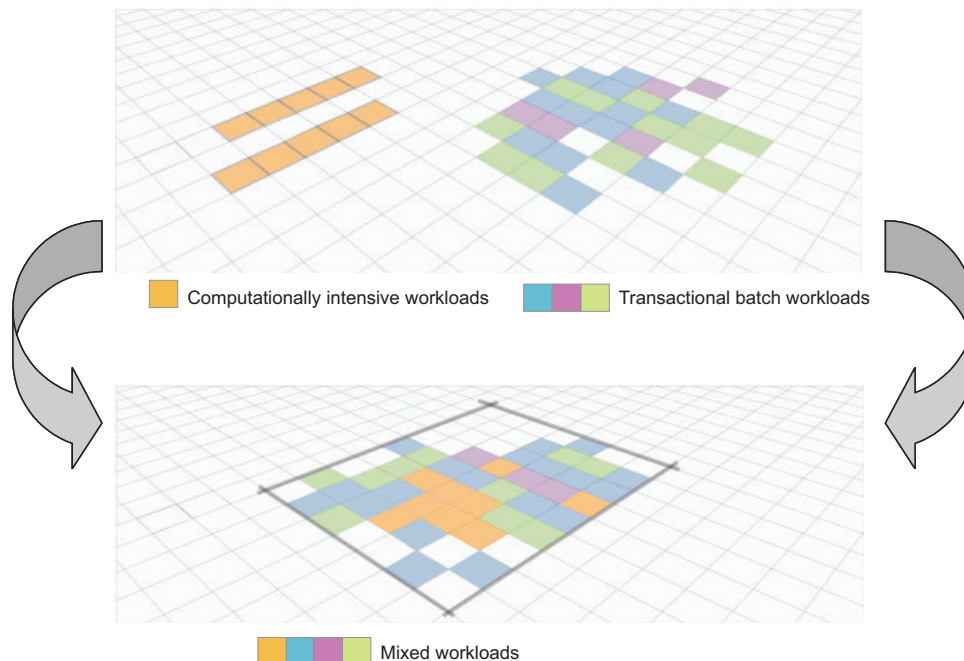
Chapter 1. Batch applications

Learn about what the batch function is, the major components, the batch environment, and the grid endpoints.

Batch overview

The Java Platform, Enterprise Edition (Java EE) applications that are typically hosted by the product perform short, lightweight, transactional units of work. In most cases, an individual request can be completed with seconds of processor time and relatively little memory. Many applications, however, must complete batch work that is computational and resource intensive.

The batch function extends the application server to accommodate applications that must perform batch work alongside transactional applications, as shown in the following graphic. Batch work might take hours or even days to finish and uses large amounts of memory or processing power while it runs.



Batch support includes a web-based application for managing jobs, called the job management console. Through this console, you can submit jobs, monitor job execution, perform operational actions against jobs, and view job logs.

Jobs express units of batch work. A job describes the work, identifies the application to perform the work, and can include additional information to help the product handle the work effectively and efficiently. Jobs are specified in an XML dialect called xJCL and can be submitted programmatically or through a command-line interface. As part of a job submission, the job is persisted in an external database and given to the job scheduler. The job scheduler distributes waiting jobs to available grid endpoints to run.

Learn about batch applications

Find links to batch resources for learning, including “How do I?...” topics, conceptual overviews, and samples.

How do I?...

Configure the batch environment and administer batch applications

- Configure the job scheduler
- Use the job scheduler and job management console to administer the batch environment
- Run batch scripts or wsadmin job scheduler commands
- Secure the job scheduler
- Configure WebSphere grid endpoints
- Configure the external scheduler interface
- Configure the unit test environment (UTE)
- Run batch jobs under user credentials
- Create and manage reports for batch statistics

Develop batch applications

- Develop COBOL container batch applications
- Generate COBOL call stubs
- Develop a simple compute-intensive application
- Develop a simple transactional batch application
- Develop a parallel job management application
- Use the batch data stream framework

Deploy batch applications

- Package EJB modules in a batch application using Rational Application Developer
- Install a batch application
- Deploy an OSGi batch application
- Submit batch jobs
- Troubleshoot batch applications

Conceptual overviews

- Batch overview
- Getting started with the batch environment
- Understanding the elements in the batch environment
- Batch applications, jobs, and job definitions
- Grid endpoints
- Unit test environment topology
- Batch jobs and their environment
- Transactional batch and compute-intensive batch programming models
- COBOL container overview

Samples

The product offers Java Batch samples. You can use these samples to explore batch capability. The samples are downloadable from the **Samples** information center. Samples include detailed deployment instructions in a readme.html file.

- xJCL samples and XML schemas for batch jobs, native jobs, and compute-intensive jobs
- WSGrid properties file examples

Getting started with the batch environment

The major components of the batch environment include the job scheduler, the interfaces to the job scheduler, the job database, and grid endpoints.

The following diagram shows the major components. You can use the command-line interface, the Enterprise JavaBeans (EJB) interface, the web services interface, and the job management console to communicate with the job scheduler. The job scheduler has a job database that contains all the jobs. The job scheduler in the diagram communicates with two node endpoints. An application server that is doing transactional work runs on another node, but does not communicate with the job scheduler. This application server is not part of the batch environment.

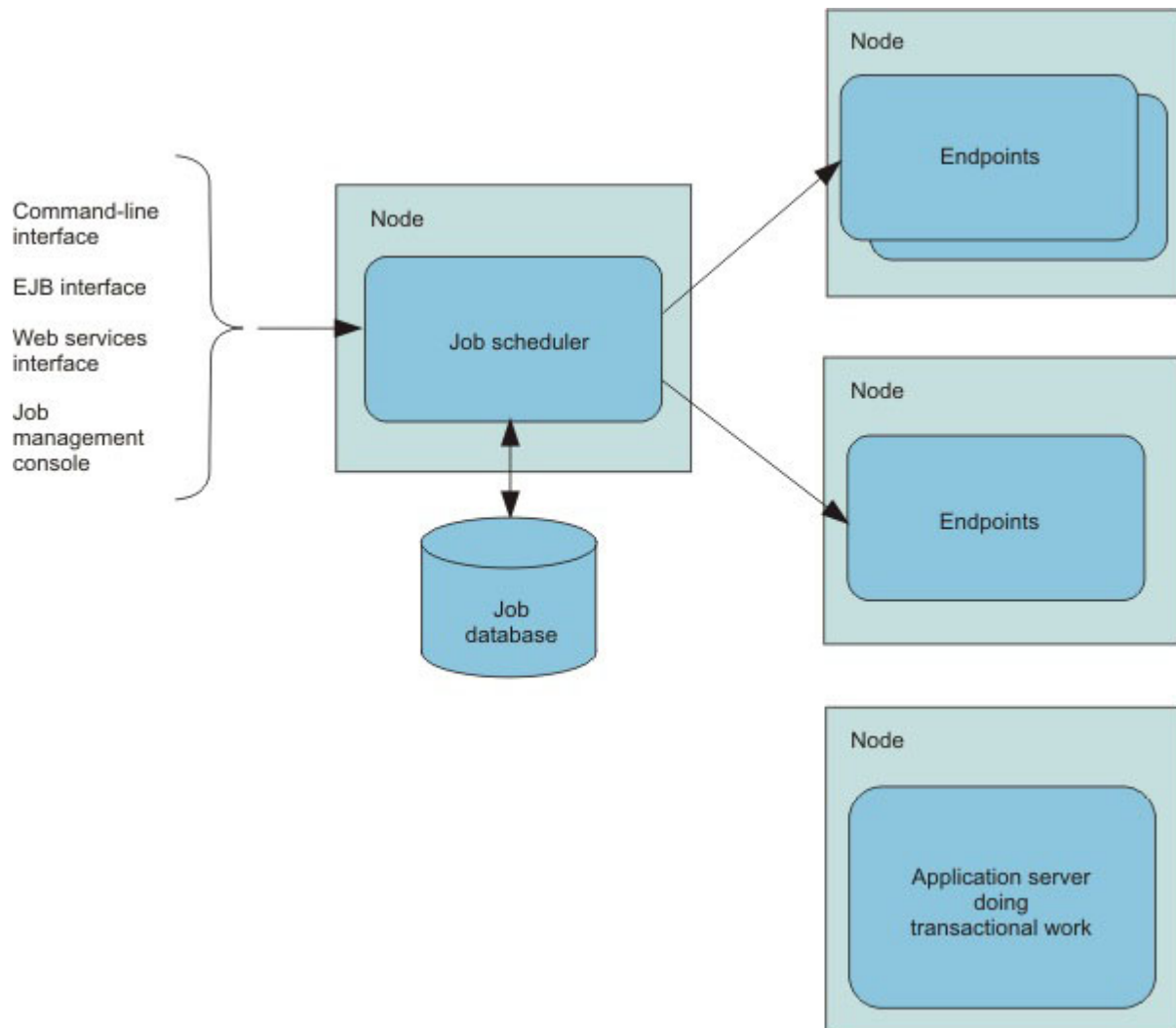


Figure 1. Batch components

The job management console provides a graphical user interface (GUI) with which you can perform job management functions. Most of the function from other interfaces is also available from the job management console.

With the command-line interface, you can submit and control the batch jobs in the system. The enterprise bean and web services interfaces provide similar function to both Java Platform, Enterprise Edition (Java

EE) and non-Java EE programs through programmatic interfaces. The administrative console provides a graphical user interface (GUI) with which you can configure the job scheduler, and view the location of endpoint servers.

Batch administrators and submitters can use the job management console to view, manage and perform job-related actions that include submitting a job, viewing of jobs, canceling or suspending a job, and resuming a suspended job.

The job scheduler accepts and schedules the execution of batch jobs. It manages the job database, assigns job IDs, and selects where jobs run.

The grid endpoints are application servers that are augmented to provide the runtime environments needed by batch applications.

- The grid endpoints support batch applications that are compute-intensive. Compute-intensive batch applications are built using a simple programming model based on asynchronous beans. Read about compute-intensive programming for more information.
- The batch system supports transactional batch applications. These applications perform record processing like more traditional Java EE applications, but are driven by batch inputs rather than interactive users. This environment builds on familiar Plain Old Java Objects (POJOs) to provide batch applications with a rich programming model that supports container-managed restartable processing and the ability to pause and cancel running jobs. Read about the batch programming model for more information.

Understanding the elements in the batch environment

A typical batch environment consists of a job scheduler, batch container, batch applications, jobs, interfaces for management functions, and database tables.

The following diagram depicts the elements of the basic batch environment:

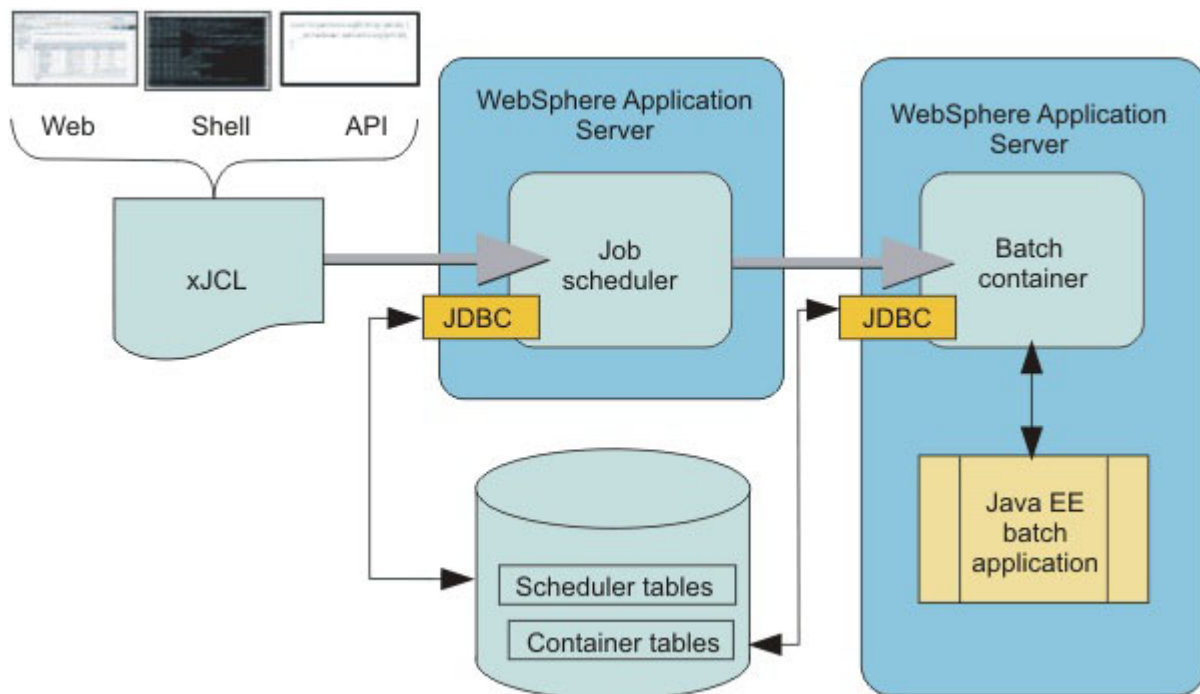


Figure 2. The batch elements

The following list describes the items in the diagram:

- **Job scheduler**

The job scheduler is the batch component that provides all job management functions, such as submit, cancel, and restart. It maintains a history of all jobs, including those waiting to run, those running, and those having already run. The job scheduler is hosted in an application server. In a Network Deployment environment, the job scheduler can also be hosted in a cluster.

- **Batch container**

The batch container is the batch component that provides the execution environment for the batch jobs. Java Platform, Enterprise Edition (Java EE) based batch applications run inside the batch container. The batch container is hosted in an application server. In a Network Deployment environment, the batch container can also be hosted in a cluster.

- **Java EE batch application**

Java EE batch applications are regular Java EE applications, deployed as Enterprise Archive (EAR) files, that contain implementations of one or more Java batch applications. These Java batch applications follow either the transactional batch or compute-intensive programming models.

- **xJCL**

Jobs are described using a job control language. The batch jobs use an XML-based job control language. The job description identifies which application to run, its inputs, and outputs.

- **Web, Shell, API**

The job scheduler exposes three API types to access its management functions: A web interface called the job management console, a shell command line called Ircmd, and APIs, available as either web services or EJBs.

- **Scheduler tables**

The job scheduler uses a relational database to store job information. It can be any relational database supported by WebSphere® Application Server. If the job scheduler is clustered, the database must be a network database, such as DB2®.

- **Container tables**

The batch container uses a relational database to store checkpoint information for transactional batch applications. The database can be any relational database supported by WebSphere Application Server. If the batch container is clustered, the database must be a network database, such as DB2.

- **JDBC**

The JDBC is standard JDBC connectivity to the scheduler and container tables, as supported by the WebSphere Application Server connection manager.

Batch applications, jobs, and job definitions

A batch application is a Java Platform, Enterprise Edition (Java EE) application that conforms to one of the batch programming models. Batch work is expressed as jobs. Jobs are made up of steps. All steps in a job are processed sequentially.

All jobs contain the following information:

- The identity of the batch application that performs the work
- One or more job steps that must be performed to complete the work
- The identity of an artifact within the application that provides the logic for each job step
- Key and value pairs for each job step to provide additional context to the application artifacts

Jobs for batch applications contain additional information specific to the batch programming model:

- Definitions of sources and destinations for data
- Definitions of checkpoint algorithms

xJCL - job definition

Jobs are expressed using an XML dialect called XML Job Control Language (xJCL). This dialect has constructs for expressing all of the information needed for both compute-intensive and batch

jobs, although some elements of xJCL are only applicable to compute-intensive or batch jobs. See the xJCL provided with the Sample applications and the xJCL schema document for more information about xJCL. The xJCL definition of a job is not part of the batch application. This definition is constructed separately and submitted to the job scheduler to run. The job scheduler uses information in the xJCL to determine where and when the job runs.

Interfaces used to submit and control jobs

xJCL jobs can be submitted and controlled through the following interfaces:

- A command-line interface
- An EJB interface described by the `com.ibm.ws.batch.JobScheduler` interface. For more information, see the API documentation for this interface.
- A web service interface
- The job management console

The grid endpoint

Batch applications run in a special runtime environment. This runtime environment is provided by a product-provided Java EE application, the batch execution environment. This application is deployed automatically by the system when a batch application is installed. The application serves as an interface between the job scheduler and batch applications. It provides the runtime environment for both compute-intensive and transactional batch applications.

Grid endpoints

The product packages and deploys batch applications as Java Platform, Enterprise Edition (Java EE) enterprise archive (EAR) files.

Deploying a batch application is like deploying a transactional Java EE application. A batch application is hosted in grid endpoints.

The deployment target is automatically enabled on the grid endpoints when you install or deploy a batch application, whether it is a compute-intensive or a batch application.

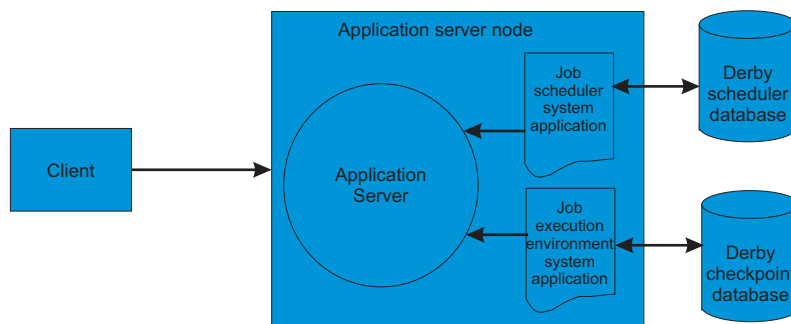
Unit test environment topology

You can use a single server unit test environment topology. The unit test environment is integrated with the product installation, and a script is provided to create the environment.

To create the environment, create a stand-alone application server. Next, use the `uteconfig.sh/.bat` script to create the batch unit test environment within that application server profile. Running the script configures the job scheduler. See topics on the `uteconfig.bat|.sh` script.

After the script is run, the unit test environment is integrated and all the necessary files are stored in the targeted profile. The selected application server is configured to host the job scheduler and job execution environment. The script also creates and configures local Derby databases to support the job components.

The following diagram shows the unit test environment topology:



Topology differences

Table 1. Differences in topology between the batch environment and the batch unit test environment. The table lists the environment and the topology.

Environment	Topology
Batch	<ul style="list-style-type: none">• Network Deployment cell• Derby database (default), DB2, Oracle
Batch unit test environment	<ul style="list-style-type: none">• Stand-alone WebSphere Application Server• Derby database (default)

Functional comparison

Table 2. Functional comparison between the batch environment and the batch unit test environment. The table indicates whether a specific function is supported in the batch unit test environment and in the batch environment.

Function	Batch unit test environment	Batch environment
Supported workload types	Batch and CI	Batch, CI, and native
Batch programming model and container	Yes	Yes
Job checkpoint / restart	Yes	Yes
Job scheduler	Yes	Yes
Job management console	Yes	Yes
Job class	Yes	Yes
Job log	Yes	Yes
High availability job scheduler	No	Yes
Scalable scheduler and container	No	Yes
Service policies	No	Yes
WLM-based scheduling	No	Yes
Job usage accounting	No	Yes

Batch frequently asked questions

When you use batch, you might have questions about the functionality of some of its features.

What are the criteria that the job scheduler uses to select a JVM to run the job?

Availability, capability, and capacity.

Can the job scheduler be run in an active-active mode? What kind of control is used to synchronize the state between them?

Yes, you can have multiple active-active job scheduler instances. For this, you must provide a network database (like DB2) for the job scheduler to persist job and job definition information.

What control does the administrator have to control batch jobs and the job scheduler from a user-interface perspective?

The product provides a job console that you can use to submit, monitor, and manage all jobs in the domain from a single location. This job management console also includes support for creating and managing job schedules (jobs that are run at a specific time or day once or repeatedly).

Chapter 2. Administering the batch environment

You can configure the batch environment and manage batch jobs.

Configuring the batch environment

Configuring the batch environment includes configuring the job scheduler and grid endpoints. The job scheduler accepts job submissions and determines where to run them. Configurations for the job scheduler includes the selection of the deployment target, datasource JNDI name, database schema name, and endpoint job log location to be configured for the schedule. Batch applications are hosted in grid endpoints.

Environment planning for transactional batch applications and compute-intensive applications

When planning your batch environment, consider certain factors that can help you design your environment to best suit your needs.

Before you build your environment, carefully consider the goals that you want to accomplish. For example, you can configure your batch environment in an existing cell or build a new cell. Also, you must decide what relational database to use, the security you need, and what your availability requirements are. The following sections contain information about each of these considerations.

New or existing cell

You can choose to configure your batch environment in an existing WebSphere Application Server cell or you can build a new cell entirely. Your choice depends on whether you want a new environment isolated from any existing WebSphere Application Server environment, or whether you want to add the capabilities of batch to an existing environment.

On the application server nodes where you want the job scheduler and batch container functions, use the administrative console to activate the functions. No action is necessary on the deployment manager node.

Job types

There are two job types. They are hosted in the WebSphere Application Server environment.

1. Transactional batch

Runs transactional batch applications that are written in Java and implement a WebSphere Application Server programming model. They are packaged as enterprise archive (EAR) files and are deployed to the batch container hosted in an application server or cluster.

The transactional batch programming model provides a container-managed checkpoint/restart mechanism that enables batch jobs to be restarted from the last checkpoint if interrupted by a planned or unplanned outage.

2. Compute-intensive

Runs compute-intensive applications that are written in Java and implement a WebSphere Application Server programming model. They are packaged as EAR files and are deployed to the batch container hosted in an application server or cluster.

The compute-intensive programming model provides a lightweight execution model based on the common framework

For all batch environments, you must deploy the job scheduler on a WebSphere Application Server server or cluster. To set up an environment to host transactional batch or compute-intensive job types, you must deploy the batch container to at least one WebSphere Application Server server or cluster. The

transactional batch, compute-intensive applications, or both are installed on the same WebSphere Application Server server or cluster.

Relational database

The job scheduler and batch container both require access to a relational database. The relational database used is JDBC connected. Access to the relational database is through the underlying WebSphere Application Server connection management facilities. The relational databases supported are the same as those relational databases supported by WebSphere Application Server, including DB2, Oracle, and others.

The simple file-based Apache Derby database is automatically configured for you by default so that you can quickly get a functioning environment up and running. However, do not use the Derby database for production use. Moreover, the default Derby database does not support a clustered job scheduler, nor a clustered batch container.

A highly available environment includes both a clustered job scheduler, and one or more clustered batch containers. Clustering requires a network database. Use production grade databases such as DB2 for this purpose. Network Derby works also, but lacks the robustness necessary for production purposes. Do not use the network version in production.

Note: Application JPA settings always override the settings on this page.

Security considerations

Security for the batch environment is based on the following techniques:

1. WebSphere authentication for access to job scheduler interfaces. Users defined to the active WebSphere security registry can authenticate and gain access to the web, command line, and programmatic interfaces of the job scheduler.
2. Role-based security for permission rights to job. Authenticated users must be assigned to the appropriate roles in order to perform actions against jobs. There are three roles:

Irsubmitter

Users in the Irsubmitter role can submit and operate on their own jobs, but on no others.

Iradmin

Users in the Iradmin role can submit jobs and operate on their own job or the jobs of anyone else.

Irmonitor

Users assigned the Irmonitor role only can view jobs and job logs of all users.

You can assign these roles using the job scheduler configuration page in the administrative console.

High availability considerations

Use clustering for high availability of batch components. Deploy and operate on clusters using the job scheduler and batch container.

Use typical application clustering techniques with the job scheduler to ensure that it is highly available. The job scheduler supports multiple methods of access to its APIs: web application, command line, web service, and Enterprise JavaBeans (EJB). Ensuring that highly available network access to a clustered job scheduler depends on which job scheduler API access method. The batch container is made highly available by deploying it to a cluster. The job scheduler automatically recognizes the batch container is clustered and takes advantage of it to ensure a highly available execution environment for the batch jobs that run there.

Configuring the unit test environment (UTE) in Rational Application Developer

Configure the unit test environment (UTE) by using the `uteconfig` script on the WebSphere Application Server profile that is used in Rational® Application Developer.

Before you begin

Your product installation must have a stand-alone application server profile; for example, `app_server_root/profiles/AppSrv01`.

About this task

You can run the `uteconfig` script to create a job scheduler configuration on the application server profile. The script creates Derby resources, deploys the job scheduler application, creates a work manager, and otherwise configures a UTE on the profile.

Procedure

1. Locate the `bin` directory of the application server profile for which you want to configure a job scheduler.

The `uteconfig.bat| .sh` script is located in the `bin` directory of the application server profile; for example, `app_server_root/profiles/AppSrv01/bin`.

The `uteconfig.bat| .sh` script is also located in the main `bin` directory of the product; for example, `app_server_root/bin`. To run the `uteconfig` script successfully from the main `bin` directory, your product installation must have only one application server profile.

2. From a command prompt open on the `bin` directory, run the `uteconfig` script.

The script takes about one minute to run and provides progress messages.

Results

The WebSphere Application Server profile now has a UTE for testing applications developed with Rational Application Developer.

Configuring the job scheduler

The job scheduler accepts job submissions and determines where to run them. As part of managing jobs, the job scheduler stores job information in an external job database. Configurations for the job scheduler includes the selection of the deployment target, data source JNDI name, database schema name, and endpoint job log location to be configured for the scheduler.

Before you begin

See the topic about creating a non-default job scheduler and grid endpoint database.

About this task

Stand-alone application servers or clusters can host the job scheduler. The first time a server or cluster is selected to host the grid scheduler, an embedded Apache Derby database is automatically created, and configured to serve as the scheduler database if the default data source JNDI name `jdbc/Trsched` is selected.

The job scheduler can be configured using the administrative console or by scripting. This topic discusses how to configure the job scheduler using the administrative console. To configure the job scheduler using the scripting language, see information on the job scheduler configuration administrative tasks.

Procedure

1. Choose the environment to host the job scheduler.

Use a stand-alone server for test environments. The stand-alone server can use the default Derby database. Use a cluster host for production environments.

Although Derby is used as the default job scheduler database, you might want to use your own database. See the topic on creating a job scheduler and grid endpoint database for more information.

2. Log on to the administrative console.
3. Click **System administration > Job scheduler** to view the Job scheduler page.
4. In the **Scheduler hosted by** list, select the deployment target.
5. Type the database schema name. The default is LRSSHEMA.
6. Select the data source JNDI name from the list. If the default of **(none)** is selected, a default embedded Derby job scheduler database is created with a value of jdbc/1rsched.
7. Type the directory where the job scheduler and the batch execution environment write the job logs. The default is `${GRID_JOBLOG_ROOT}/joblogs`.
8. Optional: Check a usage data check box.

Specifies if the scheduler records job usage data for charge-back purposes in the scheduler database.

z/OS

Specifies if job usage data for jobs are to be written in SMF.

z/OS

Job usage data can be recorded with either SMF120 subtype 20 records or SMF120 subtype 9 records. Use the RECORD_SMF_SUBTYPES job scheduler custom property to indicate the preferred subtype.

9. Click **OK** and save the configuration.
10. If administrative security is enabled, enable application security and secure the job scheduler.
See the topic on securing the job scheduler for more information. Only authorized users who are granted the Irmonitor, Irsubmitter, and Iradmin roles, or a combination of the roles, through the administrative console are allowed access to the job management console.

Job scheduler WebSphere variables

Use WebSphere variables to modify the job scheduler configuration. You can configure the amount of time that the job scheduler waits before signaling a problem with the endpoint and waits between polling the endpoint.

GRID_ENDPOINT_MISSED_HEART_BEAT_TOLERANCE_INTERVAL

Define this WebSphere variable to configure the amount of time in milliseconds that the job scheduler waits between polls of the endpoint heartbeat before signaling a problem.

Table 3. GRID_ENDPOINT_MISSED_HEART_BEAT_TOLERANCE_INTERVAL. The table includes the scope, valid values, and default for the WebSphere variable.

Scope	Valid values	Default
Cell, job scheduler node, or job scheduler server level	Time in milliseconds	5 minutes

GRID_ENDPOINT_HEART_BEAT_POLL_INTERVAL

Define this WebSphere variable to configure the amount of time in milliseconds the job scheduler waits between polls of the endpoint heartbeat.

Table 4. GRID_ENDPOINT_HEART_BEAT_POLL_INTERVAL. The table includes the scope, valid values, and default for the WebSphere variable.

Scope	Valid values	Default
Cell, job scheduler node, or job scheduler server level	Time in milliseconds	1 minute

Job scheduler System Programming Interfaces (SPI)

Use SPIs to manage a list of groups to which a user is assigned, to control user actions against jobs, to suppress the writing of log lines, and to provide an installation-specific audit string validation rule.

SPI properties file

The SPI class can be added to \$WAS_HOME/lib/classes or to the job scheduler shared library. Use the configCGSharedLib.py wsadmin script to assign the shared library to the job scheduler.

Table 5. Attributes of a property file. The table lists the property file and its attributes.

Property file	Attribute
Name	xd.spi.properties
Location	app_server_root/properties
Format	<SPI name>=<SPI implementation class>

Group membership filter SPI

Use the group membership filter system programming interface (SPI) to manage the list of groups to which a user is assigned. You can use the SPI in two ways:

- To modify the list of groups that the federated repository returns
- To serve as an alternative to the federated repository. In this case, the SPI is the source of user and group membership information that batch uses.

The SPI is called each time a user logs on to the job management console and each time a job operation is performed.

Table 6. SPI name. The following table lists the name of the SPI.

Name
group.membership.manager

Job log filter SPI

Use the job log filter SPI to suppress the writing of log lines to logs from batch applications. You can suppress writing the log lines to the server logs, job logs, or both types of logs. You can also override the application log line.

Implement the com.ibm.wsspi.batch.joblog.JobLogFilter interface by implementing the getName() method and the processJobLogLine() method. The getName() method is required for all SPI implementations. The processJobLogLine() method returns a JobLogAction object to suppress where the job log line is written. You can override the application log line with the JobLogFilterListener object. Call the JobLogFilterListener object with the updated or replaced log line and job ID that would be written to log files based on the provided JobLogAction object.

Table 7. SPI name. The following table lists the name of the SPI.

Name
spi.job.log.filter

Job log filter SPI example

```
package com.ibm.websphere.samples;
import com.ibm.websphere.grid.spi.SPI;
import com.ibm.wsspi.batch.joblog.JobLogFilter;
import com.ibm.wsspi.batch.joblog.JobLogFilterListener;
public class SampleJobLogFilter extends SPI implements JobLogFilter {
/**
 * Input:
 * jobid
 * logline - line about to be logged
 * JobLogFilterListener - call back to override logline
 * Output:
 * JobLogAction:
 * SUPPRESS - do not log this line
 * JOBLGONLY - log only to job log (not server log)
 * SERVERLOGONLY - log only to server log (not job log)
 * JOBLGSERVERLOG - log to both job log and server log
 *(this is the default action)
 */
public JobLogAction processJobLogLine(
String jobid,
String logline,
JobLogFilterListener filterListener) {
filterListener.setLogLine(jobid, "MyCompanyName:" + logline);
return JobLogAction.JOBLGONLY;
}
/**
 * Required for all Batch SPI implementations
 */
public String getName() {
return SimpleCIJobLogFilter.class.getName();
}
}
```

The processJobLogLine() method returns a JobLogAction object to suppress the writing of application log lines to the system logs. The call to the JobLogFilterListener object causes the job log lines to be appended with a standard text.

Ensure that the servers load this job log filter SPI by including a reference to this implementation class in the `app_server_root/properties/xd.spi.properties` file:

```
spi.job.log.filter=com.ibm.websphere.ci.samples.SimpleCIJobLogFilter
```

Ensure that the implementation class is available to the server through a server level shared library.

Job operation authorization SPI

The job operation authorization SPI provides administrators with further control over user actions against jobs. You can exercise fine-grained access control over each user action by permitting or denying the operation.

Invoke the SPI only after you apply the configured job security policy. The SPI is invoked only if the user is authorized to perform the operation. The SPI can override the system when the system permits the operation. However, the SPI cannot override the system when the system denies the operation. Therefore, the JobOperationAuthorizer serves only to limit the range of operations a user is authorized to perform. The SPI is not able to increase the range operations a user is authorized to perform.

Table 8. SPI name. The following table lists the name of the SPI.

Name
job.operation.authorizer

Audit string validation SPI

You can use the audit string validation SPI to provide an installation-specific audit string validation rule. You can use the validation rule to enforce local auditing requirements and provide custom error messages to guide the user to a successful save.

When the audit string validator is configured and installed, it is driven each time a repository job is saved through any of the available interfaces, which include the job management console, the command-line interface, or an API.

The audit string validator is driven and passed the name of the repository job, the current user, the xJCL, the value of the audit string, and an `AuditStringValidatorCallback` method. The audit string validator can then decide whether the audit string is valid or not. If the audit string is valid, the audit string validator returns `true`. If the audit string is not valid, the audit string validator returns `false`. If the audit string validator returns `false`, you can supply text for an error message through the `AuditStringValidatorCallback` method.

Table 9. SPI name. The following table lists the name of the SPI.

Name
audit.string.validator

Creating the job scheduler and grid endpoint database

You can create a database for the job scheduler and grid endpoint if you do not use the default Apache Derby database. The job scheduler stores job information in a relational database while the grid endpoint uses the database to track the progress of a batch job.

Before you begin

When you install the product, one Derby Java Database Connectivity (JDBC) provider is created. The Derby JDBC provider contains two data sources. One is the default Derby data source, JNDI name `jdbc/lrsched`, that points to the default Derby job scheduler database. The other, JNDI name `jdbc/pgc`, is the batch execution environment data source. If you decide to use the default data source you do not need to create the job scheduler database. The default Derby database for the job scheduler is created when the job scheduler host (deployment target) is selected through the administrative console. The default Derby database for the endpoint is created when a batch application is first installed on a node. Embedded Derby databases cannot be shared by multiple processes and are unsuitable for environments where the job scheduler must move from one node to another. For example, the job scheduler must move from one node to another in high availability scenarios.

About this task

The product supports Derby, DB2, and Oracle databases. You can use the following steps to configure the job scheduler and grid endpoint database if you decide to use a database other than the Derby database. When you create the database manually, the job scheduler and grid endpoint can use the same database.

Procedure

1. Select the correct file based on the type of database that you are going to use.

The product provides DDL files except for DB2 on the z/OS® operating system. Use the DDL files to define the job scheduler database in the `app_install_root/util/Batch` directory. The DDL files for creating the job scheduler database are named `CreateLRSCHEdTablesXxx.ddl` where `Xxx` indicates the type of database manager that the scripts are intended for. These same DDL files are used for the grid endpoint.

z/OS The product provides a SPUFI script for DB2 in the `<WAS_install_root>/util/Batch` directory. The SPUFI script is `SPFLRS`.

2. See the documentation of your database vendor for details on customizing scripts and using the database tools to run it.

What to do next

After creating the database, complete the following steps.

1. Define the XA JDBC provider for the database through the administrative console.
Consult the JDBC provider documentation for more information about defining a new JDBC provider.
2. Create the data source using the JDBC provider through the administrative console.
Define the data source at the cell level. Doing so guarantees that the database is available for each application server that hosts the job scheduler.
3. Verify that the database has been created by testing the connection on the data source that you created in the previous step.
4. Configure the job scheduler by selecting the JNDI name of the newly created data source in the job scheduler panel.
5. Specify the JNDI name of the data source that you created in a previous step as the value of the `GRID_ENDPOINT_DATASOURCE` variable.

Verifying the job scheduler installation

This topic describes how to verify that the job scheduler is installed correctly. The job scheduler is a system application and is not in the list of installed applications on the Enterprise applications page of an administrative console.

Before you begin

Privileges for the job scheduler differ, depending on the various roles. Roles include monitor, operator, configurator, and administrator. If you are a user with either a monitor or an operator role, you can only view the job scheduler information. If you have the role of configurator or administrator, you have all the configuration privileges for the job scheduler.

Procedure

1. Verify that the job scheduler is installed correctly by restarting the application server or cluster members where the job scheduler is configured.

If the application server or cluster members on which the job scheduler is installed have the started icon in the status field, the job scheduler is usually running. However, the job scheduler might have a problem and not start. You can verify whether the job scheduler started by checking the log files.

2. After the server is restarted, access the job management console through a web browser by typing `http://job_scheduler_server_host:grid_host/jmc`.

The `grid_host` port is the `WC_defaulthost` port under the server that you chose for the job scheduler. To find the `grid_host` port, go to your server in the administrative console, expand ports, and look for `WC_defaulthost`.

If you cannot access the job management console, check the appropriate log. If you specified a server in the web address, check the server log. If you specified a cluster member in the web address, check the cluster member log.

Securing the job scheduler

You can secure the job scheduler using roles, groups, or a combination of groups and roles.

Procedure

- Secure the job scheduler using roles.

A user can take a job-related action only if the user's role permits the job action. A user can be granted the `lrsubmitter` role, the `lradmin` role, or the `lrmonitor` role.

- **Distributed operating systems** Secure the job scheduler using groups on distributed operating systems.
You can secure the job scheduler using groups. A user can then act on a job only if the user and job are members of the same group.
- **Distributed operating systems** Secure the job scheduler using roles and groups on distributed operating systems.
You can secure the job scheduler using roles and groups. A user can then act on a job if the user and the job are members of the same group and the user's role permits the action.
- **z/OS** Secure the job scheduler using groups on z/OS operating systems.
You can secure the job scheduler using groups. A user can then act on a job only if the user and job are members of the same group.
- **z/OS** Secure the job scheduler using roles and groups on z/OS operating systems.
You can secure the job scheduler using roles and groups. A user can then act on a job if the user and the job are members of the same group and the user's role permits the action.

Results

You enabled security using roles, groups, or a combination of groups and roles.

What to do next

Manage jobs using the security option you selected.

Job scheduler security overview

The actions that a user can take against a job depend on the security model that is being enforced. User actions against a job can be role based, group based, or a combination of the two.

Role based security

When role-based security is enabled, you must be granted the `lrsubmitter` role, the `lradmin` role, or the `lrmonitor` role to act on a job. Users assigned the `lradmin` role have authority to perform all job scheduler application actions on all jobs regardless of job ownership. Users assigned the `lrsubmitter` role can view and act only on jobs that the submitter owns. Users assigned the `lrmonitor` role only can view jobs and job logs of all users.

Group security

In the group security model, group-affiliation alone is the basis for all job-related security decisions. The administrator does not assign job roles to specific users. A user can complete an action for a job only if the user and job are members of the same group. For example, if two users are members of the same group and each submits a job that is assigned to that same group, then both users can view and take actions against either of the two jobs.

Because WebSphere Application Server does role-based checking on job scheduler operations, you must make a single assignment of the `lradmin` role to **All Authenticated in Application's Realm**. Users in the group have the same privileges as the `lradmin` role and can perform the same operations against a job in the group that the `lradmin` role can perform.

User and group membership

The group security function requires either an implementation of the group membership SPI or a user repository that supports group membership, such as Lightweight Directory Access Protocol (LDAP), and is configured as a federated repository.

You can use the group membership filter SPI to augment the federated repository.

If you use a repository, you must configure the repository as a federated repository even if the WebSphere Application Server configuration is using only a single repository. The federated repositories function supports multiple repository technologies, including Local OS, LDAP, and Active Directory.

z/OS Additionally, the federated repositories function supports System Authorization Facility (SAF).

When you use a repository, you must define all WebSphere Application Server users and groups through the management facilities of the configured repository technology.

Group and role security

In the group and role security model, both group-affiliation and role-based security governs job-related security decisions. This means a user can take a job-related action only if the user and job are members of the same group, and the user's role permits the job action.

For example, if two users are members of the same group and each submits a job that is assigned to that same group, then both users can view and take actions against either of the two jobs, subject to their role assignments. If the first user is in the Iradmin role, that user can view and take job actions against both jobs. If the second user is in the Irsubmitter role, that user can view and take job actions only against the job that the second user submitted. If the second user is in the Irmonitor role instead of the Irsubmitter role, that user is disallowed from submitting a job, but is permitted to view jobs submitted by the first user.

Securing the job scheduler using roles

You can secure the job scheduler by mapping users and groups to specific security roles.

Before you begin

Users who are assigned the Iradmin role have the authority to perform all job scheduler application actions on all jobs regardless of job ownership, while users who are assigned with the Irsubmitter role can only act on jobs that are owned by the submitters themselves.

Users in the Irmonitor role can view and download all job logs, but cannot submit or operate on jobs.

Note: To start Ircmd.sh | .bat on an HTTPS port, you must configure SSL on the scheduler server. Following the steps in part three of the series, location in the following DeveloperWorks topic, Build Web services with transport-level security using Rational Application Developer V7, Part 3: Configure HTTPS. In order to access topics, you must be a registered user for DeveloperWorks. If you have not registered as a user for DeveloperWorks, follow the instructions on the IBM® registration page.

z/OS If you use System Authorization Facility (SAF) EJBROLE profiles on the z/OS operating system, define EJBROLE profiles for Iradmin and Irsubmitter roles. Permit these roles to the appropriate SAF user IDs. Do not control permissions through the administration console as described in the following procedure.

About this task

This sample task assumes that the job scheduler is configured. You can use the administrative console to specific security roles.

Procedure

1. Click **Security > Global security**.
2. Select administrative security and application security.

3. Configure the user account repository by specifying one of the available realm definitions.
4. After you have configured WebSphere Application Server Security, click **Apply** to save your configuration.
5. Expand **System administration > Job scheduler > Security role to user/group mapping**.
6. Select the roles to be configured.
7. Click **Look up users** if one or more users are to be assigned the target role, or click **Look up groups** if role assignment is at the group level.
8. Select the user or group to be assigned to the target role.
9. Click **OK** and save the configuration.
10. Restart the cell.

What to do next

With security enabled, provide a valid user ID and password for job actions that are performed through the command-line interface. Submit a job action through the command-line interface with the user name and password information. See the following example:

```
<app_server_root>/bin/lrcmd.[bat|sh]
-cmd=<name_of_command> <command_arguments> [-host=<host> -port=<port>]
-userid=<user_ID> -password=<password>
```

where:

- <host> is the job scheduler server host name. If not specified, the default is localhost.
- <port> is the scheduler server HTTP (HTTPS) port. If not specified, the default is 80.

See the following example:

```
D:\IBM\WebSphere\AppServer\bin\lrcmd -cmd=submit
-xJCL=D:\IBM\WebSphere\AppServer\samples\Batch\postingSampleXJCL.xml
-port=9445 -host=wasxd01.ibm.com -userid=mylradmin -password=w2g0u1tf
```

Job scheduler administrative roles and privileges

Job scheduler roles and privileges vary depending on your administrative role and the component.

Administrative roles and privileges

For definitions of administrative roles in WebSphere Application Server and how to assign them, see *Authorizing access to administrative roles*.

Table 10. Administrative roles and privileges. The table lists each component for the graphical user interface (GUI) and what privileges the component has for the monitor, operator, configurator, and administrator privileges.

GUI	Monitor privileges	Operator privileges	Configurator privileges	Administrator privileges
Job scheduler	View the information.	View the information.	Has all privileges.	Has all privileges.

Roles and privileges for securing the job scheduler

This topic describes the lradmin and lrsubmitter roles and privileges for securing the job scheduler.

Authority for different roles

You can secure the job scheduler application by enabling global security and application security. Application security secures the job management console. The job scheduler application uses a combination of both declarative and instance-based security approaches to secure jobs and commands, where only users who are assigned with the lradmin or lrsubmitter role have the authority to perform grid operations in a security-enabled environment.

As illustrated in the following table, users who are assigned with the Iradmin role have the authority to perform all job scheduler application actions on all jobs regardless of job ownership, while users who are assigned with the Irsubmitter role can only act on jobs that are owned by the submitters themselves. The **X** character represents authority in the following table.

Table 11. Authoritative roles. The table lists client commands and indicates with an **X** character whether the Iradmin role or the Irsubmitter role have authority for those commands.

Client commands	Iradmin role	Irsubmitter role
submit -xJCL=<file>	X	X
submit -job=<job name>	X	X
submit -job=<job name> -add or replace	X	N/A This is an admin command.
cancel -jobid=<jobid>	X	X (only jobs owned)
purge -jobid=<jobid>	X	X (only jobs owned)
output -jobid=<jobid>	X	X (only jobs owned)
restart -jobid=<jobid>	X	X (only jobs owned)
remove -job=<jobname>	X	N/A This is an admin command.
suspend -jobid=<jobid>	X	X (only jobs owned)
resume -jobid=<jobid>	X	X (only jobs owned)
status (showAll)	X	N/A This is an admin command.
status -jobid=<jobid>	X	X (only jobs owned)
getBatchJobRC -jobid=<jobid>	X	X (only jobs owned)
help	X	X

z/OS If you use System Authorization Facility (SAF) EJBROLE profiles on the z/OS operating system to administer role-based security, define EJBROLE profiles for Iradmin and Irsubmitter roles. Permit these roles to the appropriate SAF user IDs for batch job administrators and submitters.

Securing the job scheduler using groups on distributed operating systems

Distributed operating systems

You can secure the job scheduler using groups. A user can then act on a job only if the user and job are members of the same group.

About this task

Create a group and a user that belongs to the group. Enable group security for the job scheduler by mapping authenticated users to the Iradmin administrative security role. Assign a group to a job.

Procedure

1. Create a group and a user that belongs to that group.
Read the section on assigning users and groups to roles in the WebSphere Application Server documentation and follow the directions. For this task, an example user is user1 and an example group is BATCHGROUP.
2. Enable group security for the job scheduler.
 - a. Click **System administration > Job scheduler > Custom properties**.
 - b. Click **New** and add JOB_SECURITY_POLICY for **Name** and GROUP for **Value**.

- c. Click **Apply** to save your configuration.
- d. Click **System administration > Job scheduler > Security role to user/group mapping**.
- e. Select **Iradmin** for the role, **Map Special Subjects**, and **All authenticated in application realm**.
- f. Save the updates.
- g. Restart the server.
- h. Verify that group security is enabled.

If you see the following message in the SystemOut.log file, group security is enabled:

```
CWLRB5837I: The WebSphere Application Server Batch Feature is running under GROUP security policy.
```

3. Assign a group to a job.

A job belongs to a user group and an administrative group. If the JOB_SECURITY_ADMIN_GROUP variable is not defined, the job scheduler automatically assigns the administrative group to each job.

- Configure the value of the administrative group name through the JOB_SECURITY_ADMIN_GROUP job scheduler custom property:

```
JOB_SECURITY_ADMIN_GROUP=JSYSADMN
```

The default administrative group name is JSYSADMN.

- Assign the group using one of the following methods.

- Define the group on the group attribute in the xJCL, for example:

```
<job-name="{jobname}" group="{group-name}" ... />
```

- Set the job scheduler default group name using the JOB_SECURITY_DEFAULT_GROUP job scheduler custom property:

```
JOB_SECURITY_DEFAULT_GROUP=JSYSDFLT
```

The default group name is JSYSDFLT.

The group attribute in the xJCL takes precedence over the job scheduler custom property. If you do not specify a group name in your xJCL, the job scheduler assigns the default group name.

Results

You created a group and assigned a user to the group so that a user can manage jobs using group security.

What to do next

Manage jobs using group security.

1. Submit the job.
2. Have the user1 user that you created in a previous step act on the job, by viewing the job log, for example.

Securing the job scheduler using roles and groups on distributed operating systems

Distributed operating systems

You can secure the job scheduler using roles and groups. A user can then act on a job if the user and the job are members of the same group and the user's role permits the action.

About this task

Create a group and a user that belongs to the group. Enable group security for the job scheduler by mapping authenticated users to the Iradmin administrative security role.

Procedure

1. Create a group and a user that belongs to that group.
Read the section on assigning users and groups to roles in the WebSphere Application Server documentation and follow the directions. For this task, an example user is user2 and an example group is BATCH2GROUP.
2. Enable group and role security for the job scheduler.
 - a. Click **System administration > Job scheduler > Custom properties**.
 - b. Click **New** and add JOB_SECURITY_POLICY for **Name** and GROUPROLE for **Value**.
 - c. Click **Apply** to save your configuration.
 - d. Click **System administration > Job scheduler > Security role to user/group mapping**.
 - e. Select **Irsubmitter** for the role, **Map users...** to map the user2 user to the Irsubmitter role.
The Irsubmitter role was used for this example. You can select a different role.
 - f. Save the updates.
 - g. Restart the server.
 - h. Verify that group and role security is enabled.

If you see the following message in the SystemOut.log file, group security is enabled:

```
CWLRB5837I: The WebSphere Application Server Batch Feature is running under GROUPROLE security policy.
```

3. Assign a group to a job.

A job belongs to a user group and an administrative group. If the JOB_SECURITY_ADMIN_GROUP variable is not defined, the job scheduler automatically assigns the administrative group to each job.

- Configure the value of the administrative group name through the JOB_SECURITY_ADMIN_GROUP job scheduler custom property:

```
JOB_SECURITY_ADMIN_GROUP=JSYSADMN
```

The default administrative group name is JSYSADMN.

- Assign the group using one of the following methods.

- Define the group on the group attribute in the xJCL, for example:

```
<job-name="{jobname}" group="{group-name}" ... />
```

- Set the job scheduler default group name using the JOB_SECURITY_DEFAULT_GROUP job scheduler custom property:

```
JOB_SECURITY_DEFAULT_GROUP=JSYSDFLT
```

The default group name is JSYSDFLT.

The group attribute in the xJCL takes precedence over the job scheduler custom property. If you do not specify a group name in your xJCL, the job scheduler assigns the default group name.

Results

You created a group and a user that belongs to the group. You mapped the authenticated user to the Irsubmitter security role.

What to do next

Manage jobs using group and role security.

1. Submit the job.
2. Have the user2 user that you created in a previous step act on the job with an action that a user in the Irsubmitter role can complete.

Securing the job scheduler using groups on the z/OS operating system

z/OS

You can secure the job scheduler using groups. A user can then act on a job only if the user and job are members of the same group.

Before you begin

Start the deployment manager and all node agents.

About this task

Enable WebSphere Application Server global security. Configure the user registry bridge for federated repositories. Install and configure a VMM SAF mapping module and add the module to three login modules. Use RACF® to create a group and add a user to the group. Then assign a group to a job.

Procedure

1. Enable global security.

Read the section on enabling security in the WebSphere Application Server documentation and follow the directions. On the Global Security pages, ensure that you select the following options.

- **Enable administrative security** and **Enable application security**
- **Federated repositories** for **Available realm definitions**
If this option is not selected, select it and click **Set as current**.
- **Enable SAF Delegation** for **Authorization provider**

2. Configure the user registry bridge for federated repositories.

Read the section on configuring the user registry bridge for federated repositories using wsadmin scripting in the WebSphere Application Server documentation and follow the directions.

3. Install and configure the SampleVMMSAFMappingModule module.

Read the section on installing and configuring a custom System Authorization Facility mapping module for WebSphere Application Server and follow the directions. You add the module to the WEB_INBOUND, RMI_INBOUND, and DEFAULT login modules.

4. Synchronize your changes and restart the cell.
5. Create a group and add a user to the group.

Read the information about creating a group and adding a user to the group in the RACF user's guide, *Security Server RACF General User's Guide*.

6. Assign a group to a job.

A job belongs to a user group and an administrative group. If the JOB_SECURITY_ADMIN_GROUP variable is not defined, the job scheduler automatically assigns the administrative group to each job.

- Configure the value of the administrative group name through the JOB_SECURITY_ADMIN_GROUP job scheduler custom property:

```
JOB_SECURITY_ADMIN_GROUP=JSYSADMN
```

The default administrative group name is JSYSADMN.

- Assign the group using one of the following methods.

- Define the group on the group attribute in the xJCL, for example:

```
<job-name="{jobname}" group="{group-name}" ... />
```

- Set the job scheduler default group name using the JOB_SECURITY_DEFAULT_GROUP job scheduler custom property:

```
JOB_SECURITY_DEFAULT_GROUP=JSYSDFLT
```

The default group name is JSYSDFLT.

The group attribute in the xJCL takes precedence over the job scheduler custom property. If you do not specify a group name in your xJCL, the job scheduler assigns the default group name.

Results

You created a group and assigned a user to the group so that a user can manage jobs using group security.

What to do next

Manage jobs using group security.

1. Submit the job.
2. Have the user1 user that you created in a previous step act on the job, by viewing the job log, for example.

Securing the job scheduler using roles and groups on the z/OS operating system

z/OS

You can secure the job scheduler using roles and groups. A user can then act on a job if the user and the job are members of the same group and the user role permits the action.

Before you begin

Start the deployment manager and all node agents.

About this task

Enable WebSphere Application Server global security. Configure the user registry bridge for federated repositories. Install and configure a VMM SAF mapping module and add the module to three login modules. Then use RACF to create a group and add a user to the group. Assign a group to a job. Define EJBROLE profiles for the Iradmin and Irsupplier roles.

Procedure

1. Enable global security.

Read the section on enabling security in the WebSphere Application Server documentation and follow the directions. On the Global Security panels, ensure that you select the following options.

 - **Enable administrative security** and **Enable application security**
 - **Federated repositories** for **Available realm definitions**
 - **Enable SAF Delegation** for **Authorization provider**
2. Configure the user registry bridge for federated repositories.

Read the section on configuring the user registry bridge for federated repositories using wsadmin scripting in the WebSphere Application Server documentation and follow the directions.
3. Install and configure the SampleVMMSAFMappingModule module.

Read the section on installing and configuring a custom System Authorization Facility mapping module for WebSphere Application Server and follow the directions. You add the module to the WEB_INBOUND, RMI_INBOUND, and DEFAULT login modules.
4. Synchronize your changes and restart the cell.
5. Create a group and add a user to the group.

Read the information about creating a group and adding a user to the group in the RACF user's guide, *Security Server RACF General User's Guide*.
6. Assign a group to a job.

A job belongs to a user group and an administrative group. If the JOB_SECURITY_ADMIN_GROUP variable is not defined, the job scheduler automatically assigns the administrative group to each job.

- Configure the value of the administrative group name through the JOB_SECURITY_ADMIN_GROUP job scheduler custom property:

```
JOB_SECURITY_ADMIN_GROUP=JSYSADMN
```

The default administrative group name is JSYSADMN.

- Assign the group using one of the following methods.
 - Define the group on the group attribute in the xJCL, for example:

```
<job-name="{jobname}" group="{group-name}" ... />
```

- Set the job scheduler default group name using the JOB_SECURITY_DEFAULT_GROUP job scheduler custom property:

```
JOB_SECURITY_DEFAULT_GROUP=JSYSDFLT
```

The default group name is JSYSDFLT.

The group attribute in the xJCL takes precedence over the job scheduler custom property. If you do not specify a group name in your xJCL, the job scheduler assigns the default group name.

7. Define EJBROLE profiles for the Iradmin and Irsubmitter roles.

If you use System Authorization Facility (SAF) EJBROLE profiles on the z/OS operating system to administer role-based security, define EJBROLE profiles for the Iradmin and Irsubmitter roles. Permit these roles to the appropriate SAF user IDs for batch job administrators and submitters.

Results

You created a group and assigned a user to the group. You also permitted the user ID to the appropriate role so that the user can manage jobs if the role permits the actions.

What to do next

Manage jobs using group and role security.

1. Submit the job.
2. Have the user that you created in a previous step act on the job, by viewing the job log, for example.

Configuring WebSphere grid endpoints

This topic explains how to set up a WebSphere grid endpoint.

Procedure

1. Install a batch application on a server or cluster using the administrative console, wsadmin commands, or another supported method for deploying applications.
2. If the application is the first batch application installed on the server or cluster, restart the server or cluster.

Results

The WebSphere grid endpoints are automatically set up. By installing the application on the deployment target, the common batch container is automatically deployed on the server or cluster selected using the default Apache Derby data source jdbc/pgc. The default file-based Derby data source can be used only when using the batch function on a stand-alone application server. If you have a WebSphere Application Server Network Deployment environment, you must use a network database.

If you use the default Derby data source, no further action is required. If you use a different data source, complete the following steps to make the data source available to the WebSphere grid endpoints.

1. On the administrative console, click **Environment** > **WebSphere variables**.

2. Select the Cell scope from the list.
3. Edit the GRID_ENDPOINT_DATASOURCE variable to point to the JNDI lookup name of the job scheduler data source.
4. Save your configuration.
5. Restart all endpoints.

Endpoint WebSphere variables

Use WebSphere variables to modify the endpoint configuration. You can do such things as enable jobs to run under user credentials and configure the schema name of the grid endpoint database.

RUN_JOBS_UNDER_USER_CREDENTIAL

Define this WebSphere variable so that jobs can run under user credentials.

Table 12. RUN_JOBS_UNDER_USER_CREDENTIAL. The table includes the scope, valid values, and default for the WebSphere variable.

Scope	Valid values	Default
Cell, endpoint node, or endpoint server level	<ul style="list-style-type: none"> • true Jobs are run under user credentials • false Jobs are run under server credentials 	false

GRID_ENDPOINT_HEART_BEAT_INTERVAL

Define this WebSphere variable to configure the amount of time between heartbeat transmissions from the grid endpoint to the job scheduler.

Table 13. GRID_ENDPOINT_HEART_BEAT_INTERVAL. The table includes the scope, valid values, and default for the WebSphere variable.

Scope	Valid values	Default
Cell, endpoint node, or endpoint server level	Time in milliseconds	30 seconds

GRID_ENDPOINT_DATABASE_SCHEMA

Define this WebSphere variable to override the default database schema name for a grid endpoint. Set this variable if the grid endpoint database is different than the job scheduler database and uses a schema name other than the default of LRSSHEMA.

Table 14. GRID_ENDPOINT_DATABASE_SCHEMA. The table includes the scope, valid values, and default for the WebSphere variable.

Scope	Valid values	Default
Cell, endpoint node, or endpoint server level	Grid endpoint database name	LRSSHEMA

GRID_ENDPOINT_DATASOURCE

Define this WebSphere variable to configure the grid endpoint data source Java Naming and Directory Interface (JNDI) name.

Table 15. *GRID_ENDPOINT_DATASOURCE*. The table includes the scope, valid values, and default for the WebSphere variable.

Scope	Valid values	Default
Cell, endpoint node, or endpoint server level	Grid endpoint data source JNDI name	jdbc/pgc

GRID_MEMORY_OVERLOAD_PROTECTION

Define this WebSphere variable to enable memory-overload protection for the endpoint servers. Memory-overload protection delays the running of a job in the endpoint server if insufficient Java heap memory is available to run the job. The job is delayed until other currently running jobs complete and free up enough memory.

The endpoint server determines the amount of available memory by querying the Java virtual machine (JVM) and assessing the memory requirements of all active jobs currently running within the server.

You can specify the memory requirement for a job by defining the memory attribute of the job element in the xJCL. If you do not specify the memory attribute, then the value of the `GRID_MEMORY_OVERLOAD_PROTECTION` WebSphere variable is used as the default. If you define the `GRID_MEMORY_OVERLOAD_PROTECTION` WebSphere variable as `?`, then the endpoint server estimates the average job memory requirement by assessing the current active job count and the amount of memory currently in use.

If you do not define the `GRID_MEMORY_OVERLOAD_PROTECTION` WebSphere variable, then memory-overload protection is disabled.

Table 16. *GRID_MEMORY_OVERLOAD_PROTECTION*. The table includes the scope, valid values, and default for the WebSphere variable.

Scope	Valid values	Default
Cell, endpoint node, or endpoint server level	<ul style="list-style-type: none"> An integer value with a unit of KB (kilobytes), MB (megabytes), or GB (gigabytes). For example, specify 100MB or 25KB. A value of <code>?</code>. The endpoint server estimates the memory requirement for any job that does not define the memory attribute in the xJCL. The estimate is computed by assessing the current active job count and the amount of memory currently in use. 	None

Running batch jobs under user credentials

You can allow batch jobs to run under credentials of the user when WebSphere security is enabled.

About this task

The `RUN_JOBS_UNDER_USER_CREDENTIAL` variable allows users to enable or disable batch jobs to run under credentials of the user. When the job is dispatched to the endpoint, the batch container switches the credentials of the server to the credentials of the user. The credentials of the server are in the job step thread.





`RUN_JOBS_UNDER_USER_CREDENTIAL` can be created at any scope level and accepts values `true` or `false`. The default is `false`, which means that batch jobs run under server credentials.

When Java 2 Security is enabled, your batch applications must grant the following two permissions in the `was.policy` file of the application:

- permission `com.ibm.websphere.security.WebSphereRuntimePermission "SecOwnCredentials"`
- permission `com.ibm.websphere.security.WebSphereRuntimePermission "ContextManager.getServerCredential"`

The following steps describe how to create the custom property to enable or disable batch jobs to run under the credentials of a user after logging on to the administrative console:

Procedure

1. Click **Environment** > **WebSphere variables**.
2. Select a configuration scope, then click **New**. The general properties page opens.
3. For **Name**, type `RUN_JOBS_UNDER_USER_CREDENTIAL`.
4. For **Value**, type `True` or `False` to enable or disable jobs to run under user credential.
5. Click **OK**, then click **Save**. 
- To enable jobs to run under user credentials on z/OS, also complete step 6.
6.  Save the configuration and restart the server.  To run jobs under credentials of the user on the z/OS platform, follow these steps: 
 - a. Go to the security administration pane and click **z/OS security options**.
 - b. Enable application server and z/OS thread identity synchronization. This option specifies that application servers can process the `syncToOSThread` option for application components that specify it. Local JCA connectors might honor the MVS™ identity for authentication and authorization when an application requests a connection.
 - c. Enable the connection manager `RunAs` thread identity. This option sets the MVS identity associated with the Java Platform, Enterprise Edition (Java EE) identity on the execution thread.
 - d. Click **OK**.
 - e. Save the configuration and restart the server.

What to do next

Stop and start the server where the batch execution environment is installed.

Batch jobs and their environment

The product provides ways of managing the scheduling and execution control of background activities in a grid computing environment.

The various ways that you can manage your batch environment include using the job management console, analyzing job logs, specifying job classes, and by using classification rules.

Through the job management console, you can:

- Submit jobs
- Monitor job execution
- Perform operational actions against jobs
- View job logs
- Manage the job repository
- Manage job schedules

Job logs

A job log is a file that contains a detailed record of the execution details of a job. It is composed of both system and application messages. Job logs are stored on the endpoints where the job runs and on the application server that hosts the job scheduler.

Job logs are viewable through the job management console and from the command line.

Job classes

A job class establishes a policy for a set of batch jobs to use resources. You can control the execution time, number of concurrent jobs, job log, and job output queue storage through this policy. Each job is assigned to a job class. A default job class is provided for jobs that do not specify a class.

Job classification

Classification rules are saved in the `gridclassrules.xml` configuration file under the configuration directory of WebSphere Application Server. In batch, one `gridclassrules.xml` file exists per cell. Rules are ordered based on the priority element.

Audit string

You can successfully save jobs to the repository using an audit string. The audit string is stored in a database, but is not displayed in the job management console. You can retrieve the audit string through standard database reporting facilities. In order to provide history, the audit string is saved each time you save a job to the repository.

The database contains 1 to N versions of the audit string. N is the oldest save of the audit string and 1 is the current save of the audit string.

Job management console

The job management console is a stand-alone web interface that you can use to perform job operations such as submit, monitor, schedule, and manage.

The job management console is for managing jobs. This console provides controlled access when security is enabled.

Only authorized users who are granted the `lrsubmitter` role, the `lradmin` role, or both roles through the administrative console can be allowed access to the job management console.

When role-based security is enabled, you must be granted the `lrsubmitter` role, the `lradmin` role, or the `lrmonitor` role through the administrative console to access the job management console. When group-based security is enabled, you must be in the user group of the job or the administrative group to access the job management console.

When the security enabled is based on the group and the role, you must be in the appropriate group and the appropriate role to access the job management console. You must be in the user group of the job or the administrative group. You must also be in the `lrsubmitter` role, the `lradmin` role, or the `lrmonitor` role.

Through the job management console you can:

- Submit jobs
- Monitor job execution
- Perform operational actions against jobs
- View job logs

- Manage the job repository
- Manage job schedules

Some of the specific actions that you can execute through the job management console include the following:

- Submitting job schedules with a preferred processing time
- Configuring job schedules so that they can, for example, occur or recur on a specific time of day or week
- Choosing to delay the submission of a job by specifying the start date and time of when you want to run the job

To access the job management console:

1. Configure the job scheduler.
2. Ensure that the job scheduler is running.
If the application server or cluster members on which the job scheduler is installed have the started icon in the status field, the job scheduler is usually running. However, the job scheduler might have a problem and not start. You can verify whether the job scheduler started by checking the log files.
3. In a browser, type the web address: `http://<job scheduler server host>:<port>/jmc`.
4. If an on-demand router (ODR) is defined in the cell, type the web address: `http://<odr host>:80/jmc`.
5. If you cannot access the job management console, check the appropriate log. If you specified a server in the web address, check the server log. If you specified a cluster member in the web address, check the cluster member log.

To access the field help for the job management console, click **?** in the upper right corner of every job management panel.

Command-line interface for batch jobs

The command-line interface interacts with the job scheduler to submit and manipulate a batch job. It is located in the `app_server_root/bin` directory as the `1rcmd.sh` or `1rcmd.bat` script and can be started from any location in the WebSphere cell.

Use the `1rcmd` script to perform the following commands:

Table 17. 1rcmd commands. The table includes arguments, a description, and additional information for the 1rcmd command.

Command	Arguments	Description	Additional Information
Display usage information for <code>1rcmd</code> .	None	The command displays usage information for the <code>1rcmd</code> command.	Example: <code>1rcmd</code>

Table 17. `lrcmd` commands (continued). The table includes arguments, a description, and additional information for the `lrcmd` command.

Command	Arguments	Description	Additional Information
Submit a job to the job scheduler.	<pre>-cmd=submit -xJCL=<xjcl_filename> [-host=<host>] [-port=<port>], or -cmd=submit -job=<job_name> [-startDate=<startDate>] -startTime=<startTime>] [-host=<host>] [-port=<port>]</pre>	<p>When an XML Job Control Language (xJCL) file is specified, <code>-xJCL=<xjcl_filename></code> specifies the path of the xJCL to be submitted from the file system and optionally saved. Optional arguments:</p> <ul style="list-style-type: none"> Use <code>-job=<job_name></code> as the name of a saved XJCL in xJCL repository to specify the name to use when saving the xJCL to the repository of job xJCL. See <code>-cmd=save</code> for additional information. Use <code>-add</code> to add the xJCL to the repository of job xJCL using the specified job name. Use <code>-replace</code> to replace or add the xJCL to the repository of job xJCL using the specified job name. Use <code>-startDate=<startDate></code> as the date in which the job is submitted to run where the required <code>startDate</code> format is <code>yyyy-MM-dd</code>. Requires the <code>-startTime</code> parameter to be defined. Use <code>startTime=<startTime></code> as the time in which the job is submitted to run where the required <code>startTime</code> format is <code>HH:mm:ss</code>. This parameter requires you to define the <code>-startDate</code> parameter as well. Use <code>-host=<host></code> as the on-demand router (ODR) host name or job scheduler server host name. If not specified, the default is <code>localhost</code>. Use <code>-port=<port></code> as the ODR HTTP Proxy address or job scheduler server HTTP port. If not specified, the default is 80. <p>Both variations of the command return a job ID for the submitted job.</p>	<p>Examples:</p> <ul style="list-style-type: none"> <code>lrcmd -cmd=submit -xJCL=myxjcl.xml -host=myhost -port=81</code> <code>lrcmd -cmd=submit -xJCL=myxjcl.xml</code> <code>lrcmd -cmd=submit -job=myjob</code> <code>lrcmd -cmd=submit -xJCL=myxjcl.xml -add -job=myjob</code> <code>lrcmd -cmd=submit -xJCL=C:\myXJCL -add -job=MyJob -port=80 -startDate=2005-11-25 -startTime=23:59:00</code> <code>lrcmd -cmd=submit -job=MyJob -startDate=2005-11-25 -startTime=23:59:00</code>

Table 17. `lrcmd` commands (continued). The table includes arguments, a description, and additional information for the `lrcmd` command.

Command	Arguments	Description	Additional Information
Cancel a previously submitted job.	<code>-cmd=cancel</code> <code>-jobid=<jobid></code> <code>[-<host>] [-port=<port>]</code>	<p>This command cancels the start of a previously submitted job, or cancels the execution of a running job.</p> <p>Use <code>-jobid=<jobid></code> as the job ID assigned to the job by the job scheduler. The job ID is returned by the <code>lrcmd -cmd=submit</code> command that initially submitted the job. The <code>-cmd=status</code> command can also be used to identify the job ID for a particular job.</p> <p>Optional arguments:</p> <ul style="list-style-type: none"> Use <code>-host=<host></code> as the ODR host name or job scheduler server host name. If not specified, the default is <code>localhost</code>. Use <code>-port=<port></code> as the ODR HTTP Proxy address or job scheduler server HTTP port. If not specified, the default is 80. 	<p>Example:</p> <pre>lrcmd -cmd=cancel -jobid=myjob:2 -host=myLRShost -port=9083</pre>
Restart a job.	<code>-cmd=restart</code> <code>-jobid=<jobid></code> <code>[-host=<host>] [-port=<port>]</code>	<p>This command restarts the start of a job. Only jobs in restartable state can be restarted.</p> <p>Use <code>-jobid=<jobid></code> as the job ID assigned to the job by the job scheduler. The job ID is returned by the <code>lrcmd -cmd=submit</code> command that initially submitted the job. The <code>-cmd=status</code> command can also be used to identify the job ID for a particular job.</p> <p>Optional arguments:</p> <ul style="list-style-type: none"> Use <code>-host=<host></code> as the ODR host name or job scheduler server host name. If not specified, the default is <code>localhost</code>. Use <code>-port=<port></code> as the ODR HTTP Proxy address or job scheduler server HTTP port. If not specified, the default is 80. 	<p>Only a batch job associated with batch applications can be restarted. When a batch job is canceled using the <code>-cmd=cancel</code> command, its state is changed to restartable.</p> <p>When the job is restarted, processing resumes from the last successfully committed checkpoint.</p> <p>Example:</p> <pre>lrcmd -cmd=restart -jobid=myjob:2 -host=myLRShost -port=9081</pre>

Table 17. `lrcmd` commands (continued). The table includes arguments, a description, and additional information for the `lrcmd` command.

Command	Arguments	Description	Additional Information
Purge job information.	<code>-cmd=purge</code> <code>-job=<jobid></code> <code>[-host=<host>] [-port=<port>]</code>	<p>This command purges job information from the job scheduler and grid endpoints.</p> <p>The job scheduler maintains information about a job after the job has completed. The purge command permanently deletes job information from the job scheduler and grid endpoint databases. The command also purges the job log of the job.</p> <p>Use <code>-jobid=<jobid></code> as the job ID assigned to the job by the job scheduler. The job ID is returned by the <code>lrcmd -cmd=submit</code> command that initially submitted the job. The <code>-cmd=status</code> command can also be used to identify the job ID for a particular job.</p> <p>Optional arguments:</p> <ul style="list-style-type: none"> • Use <code>-host=<host></code> as the ODR host name or job scheduler server host name. If not specified, the default is localhost. • Use <code>-port=<port></code> as the ODR HTTP Proxy address or job scheduler server HTTP port. If not specified, the default is 80. 	<p>The job scheduler maintains information about a job after the job has completed. The purge command permanently deletes job information from the job scheduler and grid endpoint databases. The command also purges the job log of the job.</p> <p>Example:</p> <pre>lrcmd -cmd=purge -jobid=myjob:2</pre>
Save an xJCL to the job repository.	<code>-cmd=save</code> <code>-xJCL=<xjcl_filename></code> <code>-job=<job_name></code> <code>[-host=<host>] [-port=<port>]</code>	<p>This command saves an xJCL document in the job repository for use by future <code>-cmd=submit</code> commands.</p> <ul style="list-style-type: none"> • Use <code>-xJCL=<xjcl_filename></code> to specify the file name of the xJCL file to be saved. • Use <code>-job=<job_name></code> to specify the name to use when saving the xJCL to the repository of job xJCL. • The job name can be used on future <code>-cmd=submit</code> commands to reference the saved xJCL. <p>Optional arguments:</p> <ul style="list-style-type: none"> • Use <code>-host=<host></code> as the ODR host name or job scheduler server host name. If not specified, the default is localhost. • Use <code>-port=<port></code> as the ODR HTTP Proxy address or job scheduler server HTTP port. If not specified, the default is 80. 	<p>Examples:</p> <ul style="list-style-type: none"> • <code>lrcmd -cmd=save -xJCL=myxjcl.xml -job=myjob -host=myODRHost -port=82</code> • <code>lrcmd -cmd=submit -job=myjob</code>

Table 17. `lrcmd` commands (continued). The table includes arguments, a description, and additional information for the `lrcmd` command.

Command	Arguments	Description	Additional Information
Remove a job from the job repository.	<code>-cmd=remove</code> <code>-job=<job_name></code> <code>[-host=<host>] [-port=<port>]</code>	This command removes a previously saved xJCL document from the job repository. Use <code>-job=<job_name></code> to specify the name assigned to the job when you previously saved the file to the job repository. <ul style="list-style-type: none"> Use <code>-host=<host></code> as the ODR host name or job scheduler server host name. If not specified, the default is <code>localhost</code>. Use <code>-port=<port></code> as the ODR HTTP Proxy address or job scheduler server HTTP port. If not specified, the default is 80. 	Example: <code>lrcmd -cmd=remove -jobid=myjob:2 -host=myhost -port=9083</code>
Show the status of a batch job.	<code>-cmd=status</code> or <code>-cmd=status</code> <code>-jobid=<jobid></code> <code>[-host=<host>] [-port=<port>]</code>	This command displays status information about one or more jobs in the job scheduler database. Optional argument: <code>-job=<jobid></code> , if specified, indicates that only job information for the specified job is displayed.	Examples: <ul style="list-style-type: none"> <code>lrcmd -cmd=status host=myODRHost -port=83</code> <code>lrcmd -cmd=submit -xJCL=myxjcl.xml</code> (returns job ID <code>LongRunningScheduler:17</code>) <code>lrcmd -cmd=status -jobid=LongRunningScheduler:17</code>
Suspend a job.	<code>-cmd=suspend</code> <code>-jobid=<jobid></code> <code>-seconds=<seconds></code> <code>[-host=<host>] [-port=<port>]</code>	This command suspends the start of a grid batch job for the specified number of seconds. Unless manually resumed (with <code>lrcmd -cmd=resume</code> , for example), the job automatically resumes running after the specified number of seconds. Use <code>-jobid=<jobid></code> as the job ID assigned to the job by the job scheduler. The job ID is returned by the <code>lrcmd -cmd=submit</code> command that initially submitted the job. The <code>-cmd=status</code> command can also be used to identify the job ID for a particular job. Optional arguments: Use <code>-seconds=<seconds></code> to indicate the number of seconds that the job start is suspended. If not specified, the default value of 15 seconds is used. If <code>-seconds=0</code> is specified, the job does not start until manually resumed. <ul style="list-style-type: none"> Use <code>-host=<host></code> as the ODR host name or job scheduler server host name. If not specified, the default is <code>localhost</code>. Use <code>-port=<port></code> as the ODR HTTP Proxy address or job scheduler server HTTP port. If not specified, the default is 80. 	Examples: <code>lrcmd -cmd=submit -xJCL=myxjcl.xml</code> (returns job ID <code>myjob:23</code>). After job <code>myjob:23</code> has begun execution, it can be suspended for five minutes (for example), with: <code>lrcmd -cmd=suspend -jobid=myjob:23 -seconds=300 -port=81 -host=myODRHost</code> Execution of the job can be resumed before the 5 minutes expires with: <code>lrcmd -cmd=resume -jobid=myjob:23</code>

Table 17. `lrcmd` commands (continued). The table includes arguments, a description, and additional information for the `lrcmd` command.

Command	Arguments	Description	Additional Information
Resume start of a previously suspended job.	<code>-cmd=resume</code> <code>-jobid=<jobid></code> <code>[-host=<host>] [-port=<port>]</code>	This command resumes start of a previously suspended batch job. Use <code>-jobid=<jobid></code> as the job ID assigned to the job by the job scheduler. The job ID is returned by the <code>lrcmd -cmd=submit</code> command that initially submitted the job. The <code>-cmd=status</code> command can also be used to identify the job ID for a particular job.	See description of <code>-cmd=suspend</code> .
Display the output for a job.	<code>-cmd=output</code> <code>-jobid=<jobid></code> <code>[-host=<host>] [-port=<port>]</code>	Displays the output generated by the job scheduler and grid endpoint during the execution of the specified job. Use <code>-jobid=<jobid></code> as the ID assigned to the job by the job scheduler. The job ID is returned by the <code>lrcmd -cmd=submit</code> command that initially submitted the job. The <code>-cmd=status</code> command can also be used to identify the job ID for a particular job.	(none)
Display the return code of a batch job.	<code>-cmd=getBatchJobRC</code> <code>-jobid=<jobid></code> <code>[-host=<host>] [-port=<port>]</code>	Displays the overall return code produced by a grid batch job. Use <code>-jobid=<jobid></code> as the ID assigned to the job by the job scheduler. The job ID is returned by the <code>lrcmd -cmd=submit</code> command that initially submitted the job. The <code>-cmd=status</code> command can also be used to identify the job ID for a particular job. <ul style="list-style-type: none"> • Use <code>-host=<host></code> as the ODR host name or job scheduler server host name. If not specified, the default is <code>localhost</code>. • Use <code>-port=<port></code> as the ODR HTTP Proxy address or job scheduler server HTTP port. If not specified, the default is 80. 	(none)

Table 17. `lrcmd` commands (continued). The table includes arguments, a description, and additional information for the `lrcmd` command.

Command	Arguments	Description	Additional Information
Submit a recurring job request to the job scheduler .	<pre>-cmd=submitRecurringRequest -xJCL=<XML file name> -request=<name of request> -startDate=<date where first job gets submitted> -startTime=<time where job gets submitted> -interval=<time period between job submissions> [-host=<host>] [-port=<port>] or -cmd=submitRecurringRequestjob- <job_name> -request=<name of request> -startDate=<date where first job gets submitted> -startTime=<time where job gets submitted> -interval=<time period between job submissions> [-host=<host>] [-port=<port>]</pre>	<p>Submits a recurring job request to the job scheduler. The job scheduler submits a batch job against the specified xJCL at time intervals indicated by the arguments.</p> <ul style="list-style-type: none"> Use <code>xJCL=<xjcl_filename></code> to specify the path of the xJCL to be submitted from the file system and optionally saved. Use <code>-request=<name of request></code> as the unique name of the request used to identify this recurring job submission request. Use <code>-startDate=<date where first job gets submitted></code> as the date in which the first job gets submitted for start where the required <code>startDate</code> format is <code>yyyy-MM-dd</code>. Requires the <code>-startTime</code> parameter to be defined as well. Use <code>-startTime=<time where job gets submission></code> as the time in which the first job and all subsequent recurring jobs get submitted for start where the required <code>startTime</code> format is <code>HH:mm:ss</code>. This parameter requires the <code>startDate</code> parameter to be defined as well. Use <code>-interval=<time period between job submissions></code> as the time period between two job submissions for this recurring job request, where the supported time periods are daily, weekly, and monthly. <p>Optional arguments:</p> <ul style="list-style-type: none"> The name of a saved XJCL in xJCL repository. See <code>-cmd=save</code> for additional information. <code>-host=<host></code> The ODR host name or job scheduler server host name. If not specified, default is <code>localhost</code>. <code>-port=<port></code> The ODR HTTP Proxy address or job scheduler server HTTP port. If not specified, default is 80. 	<p>Examples:</p> <ul style="list-style-type: none"> <code>lrcmd :-cmd=submitRecurringRequest -xJCL=C:\myxJCL -port=81</code> <code>-request=MyMonthlyReport</code> <code>-interval=monthly</code> <code>-startDate=2006-01-02</code> <code>-startTime=23:59:00</code> <code>lrcmd -cmd=submitRecurringRequest -job=WeeklyJob -request=MyWeeklyReport -interval=weekly -startDate=2006-01-02 -startTime=23:59:00</code>

Table 17. 1rcmd commands (continued). The table includes arguments, a description, and additional information for the 1rcmd command.

Command	Arguments	Description	Additional Information
Modify an existing recurring job request.	1rcmd -cmd=modifyRecurringRequest -request=<name of request> -xJCL=<XML file name> -interval=<time period between job submissions> -startDate=<date where first job gets submitted> -startTime=<time where job gets submitted>	<p>Arguments:</p> <ul style="list-style-type: none"> Use <name of request> as the unique name of the request used to identify this recurring job submission request modification. Use <XML file name> as the name of the XML file which describes the batch job to be scheduled to start. Use <time period between job submissions> as the time period between two job submissions for this recurring job request, where the supported time periods are daily, weekly, and monthly. Use <date where first job gets submitted> as the date in which the first job gets submitted for start, where the required startDate format is yyyy-MM-dd. Requires the -startTime parameter to be defined as well. Use <time where job gets submission> as the time in which the first job and all subsequent recurring jobs get submitted to start, where the required startTime format is HH:mm:ss. Requires the -startDate parameter to be defined as well. <p>One of the following parameters must be defined for this command: -xJCL, -interval (-startDate and -startTime).</p> <p>Optional arguments:</p> <ul style="list-style-type: none"> Use -port: as the ODR proxy HTTP address or job scheduler server HTTP port. The default is 80. Use -host: as the ODR host or job scheduler host. The default is localhost. 	<p>Examples:</p> <pre>1rcmd -cmd=modifyRecurringRequest -request=MyWeeklyReport -xJCL=C:\myNewXJCL -port=80 1rcmd -cmd=modifyRecurringRequest -request=MyWeeklyReport -startDate=2006-02-02 -startTime=22:30:00 -xJCL=C:\myFebXJCL -port=80</pre>

Table 18. 1rcmd commands. The table includes arguments, a description, and additional information for the 1rcmd command.

Command	Arguments	Description	Additional information
Display usage information for 1rcmd.	None	This command displays usage information for the 1rcmd command.	Example: 1rcmd

Table 18. `lrcmd` commands (continued). The table includes arguments, a description, and additional information for the `lrcmd` command.

Command	Arguments	Description	Additional information
Stop the execution of a previously submitted job.	<code>-cmd=stop</code> <code>[-jobid=<job_id></code> <code>[-host=<host>]</code> <code>[-port=<port>]</code>	This command stops the execution of a previously submitted job when a checkpoint occurs. Use <code>-jobid=<jobid></code> as the job ID assigned to the job by the job scheduler Optional arguments: <ul style="list-style-type: none"> • Use <code>-host=<host></code> as the ODR host name or job scheduler server host name. If not specified, the default is <code>localhost</code>. • Use <code>-port=<port></code> as the ODR HTTP Proxy address or job scheduler server HTTP port. If not specified, the default is 80. 	Examples: <ul style="list-style-type: none"> • <code>lrcmd -cmd=stop -jobid=MyApp:1 -port=80 -host=myodrhst.com</code> • <code>lrcmd -cmd=stop -jobid=MyApp:1 -port=9080 -host=mygshost.com -userid=myname -password=mypassword</code>
Show the symbolic variables that are referenced in the job definition xJCL.	<code>-cmd=getSymbolicVariables</code> <code>-xJCL=<xjcl_file></code> <code>[-<host>] [-port=<port>]</code>	This command shows the symbolic variables which are referenced in the job definition xJCL. Use <code>-jobid=<jobid></code> as the job ID assigned to the job by the job scheduler Optional arguments: <ul style="list-style-type: none"> • Use <code>-xJCL=<xjcl_file></code> to specify the path of the job definition xJCL file which describes the grid job. • Use <code>-job=<job_name></code> to specify the job name, which is a key in the job repository of the job scheduler. • Use <code>-host=<host></code> as the ODR host name or job scheduler server host name. If not specified, default is <code>localhost</code>. • Use <code>-port=<port></code> as the ODR HTTP Proxy address or job scheduler server HTTP port. If not specified, the default is 80. 	Examples: <ul style="list-style-type: none"> • <code>lrcmd -cmd=getSymbolicVariables -xJCL=C:\myXJCL -port=9080 -host=mygshost.com</code> • <code>lrcmd -cmd=getSymbolicVariables -job=MyJob -port=80 -host=myodrhst.com -userid=myname -password=mypassword</code>

Table 18. `lrcmd` commands (continued). The table includes arguments, a description, and additional information for the `lrcmd` command.

Command	Arguments	Description	Additional information
Save the job log.	<code>-cmd=saveJobLog</code> <code>-jobid=<job_id></code> <code>[-host=<host>]</code> <code>[-fileName=<fileName>]</code>	<p>This command saves the job log associated with the requested job identifier to the local file system.</p> <p>Use <code>-jobid=<job_id></code> as the job ID assigned to the job by the job scheduler. The job ID is returned by the <code>lrcmd -cmd=submit</code> command that initially submitted the job. The</p> <p>Use <code>-fileName=<fileName></code> to indicate the name of a file on the local file system where the compressed job log data is to be saved. The file is replaced if it exists. The file name <code><fileName></code> might not contain embedded blanks.</p> <p>Optional arguments:</p> <ul style="list-style-type: none"> Use <code>-host=<host></code> as the ODR host name or job scheduler server host name. If not specified, the default is <code>localhost</code>. Use <code>-port=<port></code> as the ODR HTTP Proxy address or job scheduler server HTTP port. If not specified, the default is 80. 	<p>Examples:</p> <ul style="list-style-type: none"> <code>lrcmd -cmd=saveJobLog -jobid=MyApp:1 -fileName=/tmp/myZippedJobLog -port=80 -host=myodrhost.com</code> <code>lrcmd -cmd=saveJobLog -jobid=MyApp:1 -fileName=/tmp/mySavedJobLog -port=9080 -host=mygshost.com -userid=myname -password=mypassword</code>
Get job log.	<code>-cmd=getJobLog -jobid=<job_id></code>	<p>Displays the job log associated with the requested job identifier.</p> <p>Use <code>-jobid=<job_id></code> as the job ID assigned to the job by the job scheduler. The job ID is returned by the <code>lrcmd -cmd=submit</code> command that initially submitted the job.</p> <p>Optional arguments:</p> <ul style="list-style-type: none"> Use <code>-host=<host></code> as the ODR host name or job scheduler server host name. If not specified, the default is <code>localhost</code>. Use <code>-port=<port></code> as the ODR HTTP Proxy address or job scheduler server HTTP port. If not specified, the default is 80. 	<p>Examples:</p> <ul style="list-style-type: none"> <code>lrcmd -cmd=getJobLog -jobid=MyApp:1 -port=80 -host=myodrhost.com</code> <code>lrcmd -cmd=getJobLog -jobid=MyApp:1 -port=9080 -host=mygshost.com -userid=myname -password=mypassword</code>

Table 18. `lrcmd` commands (continued). The table includes arguments, a description, and additional information for the `lrcmd` command.

Command	Arguments	Description	Additional information
Purge job log	<code>-cmd=getJobLog -jobid=<job_id></code> <code>-logTimeStamp=<logTimeStamp></code>	Removes the job log associated with the requested job identifier and log time stamp. A job log entry remains in, for example: <code>/opt/IBM/WebSphere/AppServer/profiles/scheduler/joblogs/PostingsSampleEar_99/14022007_164535/part.0.log</code> . The entry tracks the reason why the job log was removed. <ul style="list-style-type: none"> Use <code>-jobid=<job_id></code> as the job ID assigned to the job by the job scheduler. The job ID is returned by the <code>lrcmd -cmd=submit</code> command that initially submitted the job. Use <code>-logTimeStamp=<logTimeStamp></code> to indicate the time stamp, the subdirectory name, which identifies the job log to be removed. The time stamp is returned by <code>-cmd=getLogMetaData</code>. Use <code>-userid=<user_id></code> to specify the user ID required when the job scheduler server is running in secure mode. Use <code>-password=<password></code> to specify the password required when the job scheduler server is running in secure mode. 	Examples: <ul style="list-style-type: none"> <code>lrcmd -cmd=getLogMetaData -jobid=PostingsSampleEar:99 -port=80 -host=mydrhost.com -userid=myname -password=mypassword</code> <code>lrcmd -cmd=purgeJobLog -jobid=PostingsSampleEar:99 -port=80 -logTimeStamp=14022007_164535 -host=mydrhost.com -userid=myname -password=mypassword</code>
Display the job log metadata for the requested job identifier.	<code>-cmd=getLogMetaData -jobid=<job_id></code>	The job log metadata indicates the log time stamps associated with the requested job identifier. The metadata or time stamp identifies a unique instance of the job. Logs from multiple different jobs with the same job number can exist. <p>Use <code>-jobid=<job_id></code> as the job ID assigned to the job by the job scheduler. The job ID is returned by the <code>lrcmd -cmd=submit</code> command that initially submitted the job.</p>	Examples: <ul style="list-style-type: none"> <code>lrcmd -cmd=getLogMetaData -jobid=MyApp:1 -port=80 -host=mydrhost.com</code> <code>lrcmd -cmd=getLogMetaData -jobid=MyApp:1 -port=9080 -host=mygshost.com</code>
Display the job log part list.	<code>-cmd=getLogPartList -jobid=<job_id></code> <code>-logTimeStamp=<logTimeStamp></code>	Displays the job log part list associated with the requested job identifier and log time stamp. Use the command <code>getLogMetaData</code> to return a timestamp to use with <code>-logTimeStamp=<timestamp></code> . <p>Use <code>-jobid=<job_id></code> as the job ID assigned to the job by the job scheduler. The job ID is returned by the <code>lrcmd -cmd=submit</code> command that initially submitted the job.</p>	Examples: <ul style="list-style-type: none"> <code>lrcmd -cmd=getLogPartList -jobid=MyApp:1 -logTimeStamp=20102006_155529 -port=80 -host=mydrhost.com</code> <code>lrcmd -cmd=getLogPartList -jobid=MyApp:1 -logTimeStamp=20102006_155529 -port=9080 -host=mydrhost.com -userid=myname -password=mypassword</code>

Table 18. 1rcmd commands (continued). The table includes arguments, a description, and additional information for the 1rcmd command.

Command	Arguments	Description	Additional information
Display the job log part.	-cmd=getLogPart -jobid=<job_id> -logTimeStamp=<logTimeStamp> -logPart=<logPart>	Displays the job log part associated with the requested job identifier, log time stamp, and log part. Use -jobid=<job_id> as the job ID assigned to the job by the job scheduler. The job ID is returned by the 1rcmd -cmd=submit command that initially submitted the job. Use -logTimeStamp=<logTimeStamp>to indicate the time stamp (the subdirectory name), which identifies the job log whose part list information is to be returned. The time stamp is returned by -cmd=getLogMetaData. Use -logPart=<logPart> to indicate the portion of the job log associated with the requested job identifier and time stamp to be returned. The log part information is returned by -cmd=getLogPartList.	Examples: <ul style="list-style-type: none">• 1rcmd -cmd=submit -xJCL=myxjcl.xml -host=myhost -port=80 (returns a job identifier of PostingsSampleEar:99)• 1rcmd -cmd=getLogMetaData -jobid=PostingsSampleEar:99 (returns the timestamp 14022007_164535)• 1rcmd -cmd=getLogPart -jobid=PostingsSampleEar:99 -logTimeStamp=14022007_164535 -logPart=part.1.log
Display the size of the job log associated with the requested job identifier.	-cmd=getLogSize -jobid=<job_id> -logTimeStamp=<logTimeStamp>	This command returns the size of the job login bytes. Use -jobid=<job_id> as the job ID assigned to the job by the job scheduler. The job ID is returned by the 1rcmd -cmd=submit command that initially submitted the job. Use -logTimeStamp=<logTimeStamp> to indicate the time stamp; that is, the subdirectory name, which identifies the job log whose part list information is to be returned. The time stamp is returned by -cmd=getLogMetaData.	Examples: <ul style="list-style-type: none">• 1rcmd -cmd=getLogSize -jobid=MyApp:1 -logTimeStamp=20102006_155529 -port=80 -host=myodrhst.com• 1rcmd -cmd=getLogSize -jobid=MyApp:1 -logTimeStamp=20102006_155529 -port=9080 -host=myodrhst.com -userid=myname -password=mypassword

Table 18. 1rcmd commands (continued). The table includes arguments, a description, and additional information for the 1rcmd command.

Command	Arguments	Description	Additional information
Return the age of the job log in the seconds since it was last modified.	-cmd=getLogAge -jobid=<job_id> -logTimeStamp=<logTimeStamp>	Displays the age of the <ul style="list-style-type: none"> 1rcmd -cmd=getLogAge -jobid=MyApp:1 -logTimeStamp=20102006_155529 -port=80 -host=myodrhst.com 1rcmd -cmd=getLogAge -jobid=MyApp:1 -logTimeStamp=20102006_155529 -port=9080 -host=myodrhst.com -userid=myname -password=mypassword Job log associated with the requested job identifier. Use -jobid=<jobid> as the ID assigned to the job by the job scheduler. The job ID is returned by the 1rcmd -cmd=submit command that initially submitted the job. The -cmd=status command can also be used to identify the job ID for a particular job. Use -logTimeStamp=<logTimeStamp> to indicate the time stamp; that is, the subdirectory name, which identifies the job log whose part list information is to be returned. The time stamp is returned by -cmd=getLogMetaData.	Examples: <ul style="list-style-type: none"> 1rcmd -cmd=getLogAge -jobid=MyApp:1 -logTimeStamp=20102006_155529 -port=80 -host=myodrhst.com 1rcmd -cmd=getLogAge -jobid=MyApp:1 -logTimeStamp=20102006_155529 -port=9080 -host=myodrhst.com -userid=myname -password=mypassword

Example of retrieving output of a batch job:

```
1rcmd -cmd=output -jobid=mybatchjob:63 -host=myLRSHost -port=9081
```

Example results:

```
CWLRB4940I: com.ibm.websphere.batch.wsbatch : -cmd=output -jobid=mybatchjob:63
CWLRB5000I: Wed Jun 15 17:55:36 EDT 2005 : com.ibm.websphere.batch.wsbatch : response to output
CWLRB1740I: [Wed Jun 15 17:55:36 EDT 2005] Job [mybatchjob:63] is in job setup.
CWLRB1760I: [Wed Jun 15 17:55:37 EDT 2005] Job [mybatchjob:63] is submitted for execution.
CWLRB2420I: [Wed Jun 15 17:55:37 EDT 2005] Job [mybatchjob:63] Step [Step1] is in step setup.
CWLRB2440I: [Wed Jun 15 17:55:38 EDT 2005] Job [mybatchjob:63] Step [Step1] is dispatched.
CWLRB2460I: [Wed Jun 15 17:55:38 EDT 2005] Job [mybatchjob:63] Step [Step1] is in step breakdown.
CWLRB2600I: [Wed Jun 15 17:55:38 EDT 2005] Job [mybatchjob:63] Step [Step1] completed normally rc=0.
CWLRB2420I: [Wed Jun 15 17:55:39 EDT 2005] Job [mybatchjob:63] Step [Step2] is in step setup.
CWLRB2440I: [Wed Jun 15 17:55:39 EDT 2005] Job [mybatchjob:63] Step [Step2] is dispatched.
CWLRB2460I: [Wed Jun 15 17:55:40 EDT 2005] Job [mybatchjob:63] Step [Step2] is in step breakdown.
CWLRB2600I: [Wed Jun 15 17:55:40 EDT 2005] Job [mybatchjob:63] Step [Step2] completed normally rc=4.
End
```

Job logs

A job log is a file that contains a detailed record of the execution details of a job. System messages from the batch container and output from the job executables are collected. By examining job logs, you can see the life cycle of a batch job, including output from the batch applications themselves.

A job log is composed of the following three types of information:

1. xJCL - A job log contains a copy of the xJCL used to run the job, including xJCL substitution values.

2. System messages - A set of system messages that communicate the major life cycle events corresponding to the job. The following system events are recorded in a job log:
 - Begin and end of a job
 - Begin and end of a step
 - Begin and end of a checkpoint
 - Open, close, and checkpoint of a batch data stream
 - Checkpoint algorithm invocation / results
 - Results algorithm invocation / results
3. Application messages - A set of messages written to standard out and standard error by a job step program.

Job logs are viewable through the job management console. Since information is added dynamically to the job log while the job is running, you can view the latest information by selecting Refresh from the job log view. Jobs logs are viewable only if the owning scheduler is active. In addition, if the endpoint running the job is unavailable, a partial job log is the result.

Output of a job log

Job log output is collected on the job scheduler node, and on the grid execution endpoint node. The output is collected in a directory which has the format:

```

${GRID_JOBLOG_ROOT}/joblogs/<job-directory>/<timeStamp-directory>

```

where

`/${GRID_JOBLOG_ROOT}/joblogs` - The base directory for all job logs on the node. It is configurable through the endpoint job log location attribute of the job scheduler panel from the administration console. The default value for `/${GRID_JOBLOG_ROOT}` is `/${user.install.root}`.

`<job-directory>` Is generated at run time from the job name. For example, if the job ID assigned by the job scheduler is `PostingsSampleEar:99`, then the generated directory name is `PostingsSampleEar_99`

`<timeStamp-directory>` - Is generated at run time from the current date. It is in the format `ddmmyyy_hhmmss`, where `dd` is the day of the month, `mm` is a month (00 - 11), and `yyyy` is the year. `hh` is the hour of the day (00 - 23), `mm` is the minute of the hour (00 - 59) and `ss` is the seconds of the minutes (00 - 59). For example, a timestamp directory with the name `14022007_164535` means that the job began processing on 14 Mar 2007, at 16:45:35.

For example, job output from job `PostingsSampleEar:99` might be collected in the directory `/opt/IBM/WebSphere/AppServer/profiles/scheduler/joblogs/PostingsSampleEar_99/14022007_164535`.

Output on the scheduler node contains an echo of the job xJCL (before and after symbolic variable substitution, if any, is performed) and job dispatch information. Job log output from the job scheduler is collected in the job log directory in the file named `part.0.log`. Output on the execution endpoint node contains both application output and grid endpoint runtime messages. This output includes any application generated output directed to the `System.out` and `System.err` output streams. Job log output from the grid endpoint is collected in the job log directory in files with names such as `part.1.log` and `part.2.log`. However, if the job scheduler and grid endpoint are installed on the same application server, job log output from both the scheduler and the grid endpoint is collected in the job log directory in the file named `part.0.log`. Each of the log parts contains approximately 1000 records. The following example shows the contents of `part.1.log`:

```

System.out: [03/13/07 08:25:32:708 EDT] Tue Mar 13 08:25:32 EDT 2007: SimpleCI application starting...
System.out: [03/13/07 08:25:32:708 EDT] -->Will loop processing a variety of math functions for approximately 30.0 seconds!
System.out: [03/13/07 08:26:02:752 EDT] Tue Mar 13 08:26:02 EDT 2007: SimpleCI application complete!
System.out: [03/13/07 08:26:02:753 EDT] -->Actual Processing time = 30.043 seconds!
CWLRB5764I: [03/13/07 08:26:03:069 EDT] Job SimpleCEar:44 ended

```

Job classes

Job classes specify limits for resource consumption by batch jobs. A job class establishes a policy for resource consumption by a set of batch jobs. Through this policy, execution time, number of concurrent jobs, job log, and job output queue storage can be controlled. This topic lists the limits enforced by job classes.

Job classes can be configured using the administrative console and stored in an `.xml` file called `jobclass.xml` under a `profile_root/config/cells/cell_name/gridjobclasses/` directory. Each job is assigned to a job class.

A job class establishes policy for:

Execution time

Maximum time a job can run before being automatically canceled by the system.

Concurrent jobs

Maximum number of concurrently dispatched jobs of a given job class.

Job log retention

Specifies the rule for deleting aged job logs. Retention can be specified by either space or time:

Space Specified in megabytes. Job logs of the specified class are deleted from oldest to newest on an endpoint if the sum of space used by job logs exceeds the specified maximum.

Time Specified as an integral number of days. Job logs of the specified class older than N days old are automatically deleted by the system.

Job output queue

Specifies the rule for deleting jobs on the job output queue. A job is on the output queue after it has either completed, or stopped, or canceled. Output queue policy allows for automatic purging of the output queue by:

Number

Specified as an integral number of jobs. When jobs on the output queue of the specified class exceed this number, they are deleted oldest to newest until the total is less than the specified number.

Time Specified as an integral number of days. Job logs of the specified class older than N days old are automatically deleted by the system.

Following are the limits enforced by job classes:

maxExecutionTime

An integer, which specifies the maximum number of seconds a job is allowed to run before it is canceled.

maxConcurrentJob

An integer, which specifies the maximum number of jobs belonging to same job class that can be dispatched to a cell. When this limit is reached, new jobs belonging to the same job class are not dispatched until the ones that are currently running complete execution.

maxClassSpace

An integer that specifies the amount of space, in megabytes, that is allowed for a job log belonging to this job class. When this limit is reached, job logs are deleted oldest to newest.

maxFileAge

An integer, which specifies the number of days a job log of this job class is stored. Job logs older than the number of days are deleted.

maxJob

An integer, which specifies the maximum number of jobs of this class that are allowed on the output queue. When this limit is exceeded, the job is automatically purged, oldest to newest.

maxJobAge

An integer value which specifies the maximum of number of days a job of this class is allowed on the output queue. Jobs older than this value are automatically purged.

Creating and managing reports for batch statistics

You can generate reports to view statistics of the job scheduler and endpoints.

Before you begin

Reports are charts that show runtime data. You must configure your environment, including servers, clusters, and applications to display data in the charts.

About this task

You can use the administrative console to generate reports and view statistics of the job scheduler and endpoints.

Procedure

1. In the administrative console, click **Runtime operations > Reports** and go to the **Reports** tab.
2. Add a new chart.
Click **Open a New Chart Tab**. A new tab opens with a blank chart.
3. Specify the scope for the chart from which the charted data is derived.
 - a. Click **Change Scope** to specify the scope.
 - b. For **Object type**, select a node, dynamic cluster, cluster, or application server scope.
 - c. For **Object instance**, select an object instance of the scheduler or endpoints.
 - d. Click **OK**.
4. Set the data set type to **Use current scope as the data set**.
 - a. Click **Add Data**.
 - b. For **Data Set Type**, select **Use current scope as the data set**.
5. Choose metrics from the selected data set to add to the chart.

Table 19. Scheduler. The table lists each metric for the scheduler followed by a description.

Metric	Description
Jobs queued	Number of jobs that are queued at the scheduler
Jobs dispatched	Number of jobs that are dispatched at the scheduler
Jobs error	Number of dispatch errors that occurred for jobs
Jobs queue time	The average time in milliseconds that a job spent in the queue
Jobs dispatch time	The average time in milliseconds that a job spent being dispatched
Jobs dispatch error time	The average time in milliseconds that a job spent being dispatched when dispatch error occurred

Table 20. Endpoints. The table lists each metric for the endpoints followed by a description.

Metric	Description
Jobs requested	Number of jobs which arrive at the execution environment (endpoint application) for processing.
Jobs completed	Number of jobs which run to completion at the execution environment.
Jobs execution time	The average time in milliseconds that a job spends running.

6. After you add data, click **OK**.

What to do next

Generate and view the report.

Job scheduler integration with external schedulers

You can integrate the job scheduler with an external workload scheduler by configuring and securing the job scheduler, enabling the interface, and running batch jobs with the WSGRID utility.

See the following topics to learn more:

- “Integration of an external workload scheduler to manage batch workloads”
- For configuring an external scheduler interface, see “Configuring the external scheduler interface” on page 47.
- For information about how to run batch jobs with the WSGRID utility, see “WSGRID command-line utility” on page 249

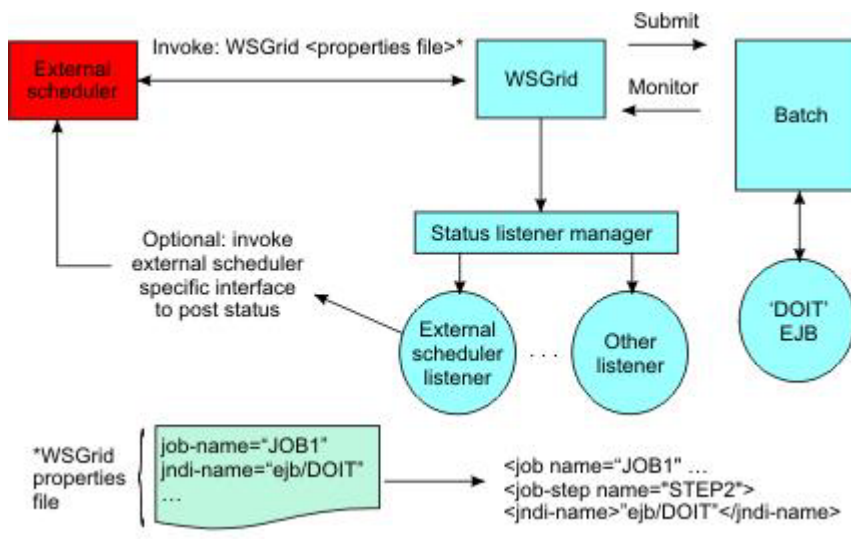
Integration of an external workload scheduler to manage batch workloads

Many customers already use an external workload scheduler to manage batch workloads on the z/OS operating system. While Java batch running inside a WebSphere Application Server environment is attractive, a way to control batch jobs through an external workload scheduler is important.

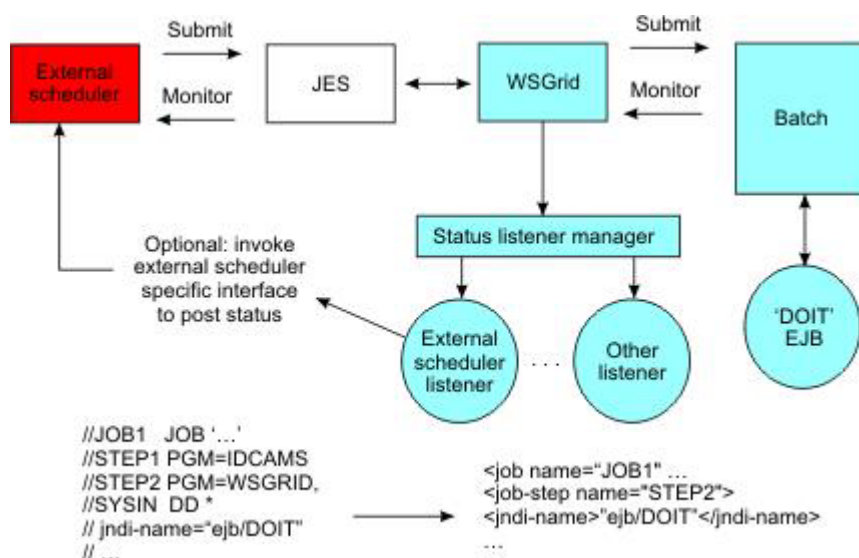
External scheduler integration

Since an external scheduler does not know how to directly manage batch jobs, a proxy model is used. The proxy model uses a regular JCL job to submit, monitor, or submit and monitor the batch job. The JCL job step invokes a special program provided by batch, named WSGRID. The WSGRID application submits and monitors a specified batch job. WSGRID writes intermediary results of the job into the JCL job log. WSGRID does not return until the underlying job is complete, consequently providing a synchronous execution model. Since the external scheduler can manage JCL jobs, it can manage a JCL job that invokes WSGRID. Using this pattern, the external scheduler can indirectly manage a job. An optional plug-in interface in the job scheduler enables a user to add code that updates the external scheduler operation plan to reflect the unique state of the underlying job, such as job started, step started, step ended, job ended. The WSGRID program is written with special recovery processing so that if the JCL job is canceled, the underlying job is canceled also, thus ensuring synchronized life cycle of the two jobs.

Distributed operating systems **z/OS** The following diagram shows the job control by an external workload scheduler, but without JES being required.



z/OS The following diagram shows the job control by an external workload scheduler for the z/OS platform environment. In this diagram, the Tivoli® Workload Scheduler is shown as an example workload scheduler.



Configuring the external scheduler interface

You can configure an external scheduler interface to control the workload for batch jobs.

Before you begin

Ensure that you configure and secure the job scheduler.

About this task

You can set up the external scheduler interface by using the default messaging provider as a Java Message Service (JMS) provider.

z/OS You also have the option of setting up the external scheduler interface by using WebSphere MQ as a messaging provider.

Procedure

- Set up the external scheduler interface using the default messaging provider.
- **z/OS** Set up the external scheduler interface using WebSphere MQ.

Setting up the external scheduler interface using the default messaging provider

The external scheduler interface uses Java Message Service (JMS) as its default messaging provider. JMS is a bidirectional communication mechanism between an external client and the job scheduler.

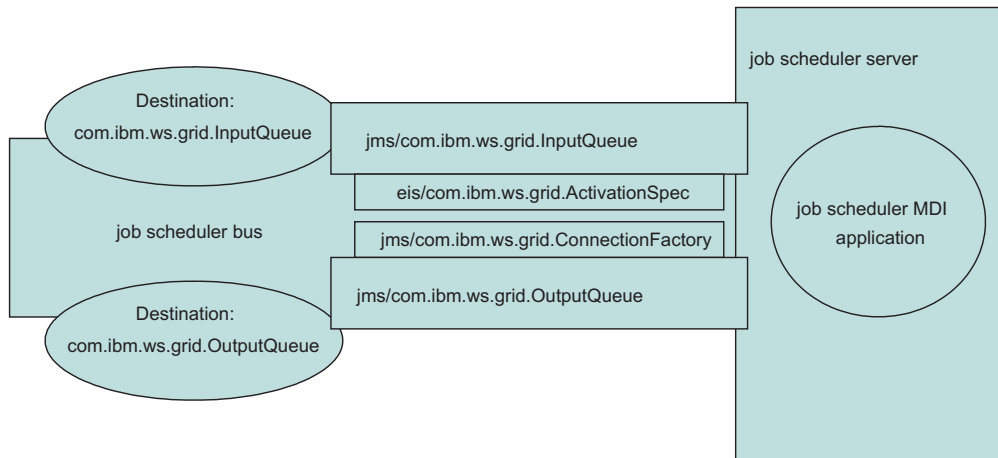
Before you begin

Ensure that you configure and secure the job scheduler.

About this task

You can run the `wsgriidConfig.py` script to enable the external scheduler interface.

The following diagram illustrates the elements that the script performs to set up the external scheduler interface. These elements include the job scheduler server, the job scheduler bus, the `jms/com.ibm.ws.grid.InputQueue` JMS input queue, and the `jms/com.ibm.ws.grid.OutputQueue` output queue:



Procedure

1. Run the `wsgriidConfig.py` script to enable the external scheduler interface.

By running the script from the deployment manager, you configure the job scheduler message-driven interface, the service integration bus and the JMS queues. For example:

```
wsadmin -user <username> -password <userpassword>
-f ../bin/wsgriidConfig.py -install -node <nodeName>
-server <serverName>
-providers "<hostname>,<SIB_ENDPOINT_ADDRESS>"
```

OR

```
wsadmin -user <username> -password <userpassword>
-f ../bin/wsgriidConfig.py -install -cluster <clusterName>
-providers "<hostname>,<SIB_ENDPOINT_ADDRESS>;
<hostname>,<SIB_ENDPOINT_ADDRESS>;...;
<SIB_ENDPOINT_ADDRESS>;<hostname>,<SIB_ENDPOINT_ADDRESS>"
```

For more information about the command options, read the `wsgriidConfig.py` topic.

2. If security is enabled on the administrative console, update the bus that was created.
 - a. On the administrative console, click **Service integration > Buses**; select the bus that was created in the previous step.
The default bus name is `JobSchedulerBus`.
 - b. Click **Security**.
 - c. Clear **Enable bus security**.
 - d. For **Permitted transports**, select **Allow the use of all defined transport channel chains**.
 - e. Click **Apply** and save the changes.
3. Change values for the CSIV2 inbound and outbound communications.
 - a. On the administrative console, click **Security > Global security > RMI/IIOP security > CSIV2 inbound communications**.
 - b. For **Transport**, select **SSL-supported** instead of **SSL-required**.
 - c. Click **Apply** and save the changes.

- d. On the administrative console, click **Security > Global security > RMI/IIOP security > CSiv2 outbound communications**.
 - e. For **Transport**, select **SSL-supported** instead of **SSL-required**.
 - f. Click **Apply** and save the changes.
4. Restart the cell for configuration changes to take effect.

What to do next

Submit a job from the external job scheduler interface to batch.

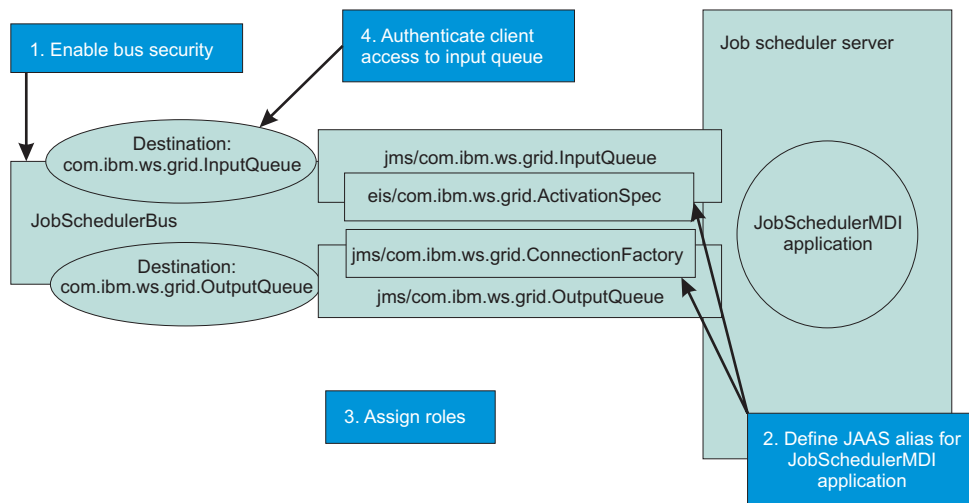
Securing the external scheduler interface when using default messaging

Securing the external scheduler interface requires securing the JobSchedulerMDI system application and the JMS resources it uses.

About this task

The following diagram shows the actions required and the environmental artifacts to which they apply. The steps in the diagram are the steps in the procedure.

Securing Job Scheduler Message-Driven Interface



The following steps show you how to secure the external scheduler interface:

Procedure

1. Enable security for the job scheduler bus in the administrative console.
 - a. Select **bus_name > Bus security > bus_name**.
 - b. Check the **Enable bus security** check box.
 - c. Click **OK**, then **Save** your configuration.
2. Define a JAAS alias.

The JMS activation specification for the JobSchedulerMDI application requires a JAAS alias. The user ID and password defined to this alias represents access to the job scheduler inbound JMS queue, `com.ibm.ws.grid.InputQueue`. The JobSchedulerMDI application also uses the JAAS alias programmatically for authenticating to the outbound queue that the job scheduler uses to communicate with its clients. The outbound queue is `com.ibm.ws.grid.OutputQueue`. Define the JAAS alias in the administrative console:

- a. Select **Security > Global security > Java Authentication and Authorization Service > J2C authentication data > New**.

- b. Define the JAAS alias.
Give the JAAS alias a name of your choice. Specify a user ID and password that provides access to the job scheduler inbound JMS queue, `com.ibm.ws.grid.InputQueue`, and enables authentication to the outbound queue, `com.ibm.ws.grid.OutputQueue`.
 - c. Click **OK** and then **Save** to save your configuration.
 - d. Assign the JAAS alias to the `activationSpec`, `com.ibm.ws.grid.ActivationSpec`.
3. Set an inter-engine authentication alias.
 - a. Select **Service integration > Buses > bus_name**.
 - b. From the inter-engine authentication alias list, select the JAAS alias that you defined in the previous step.
 - c. Click **OK**, then **Save** your configuration.
 4. Set a container-managed authentication alias.
 - a. Select **Resources > Resource Adapters > J2C connection factories > com.ibm.ws.grid.ConnectionFactory**.
 - b. From the container-managed authentication alias list, select the JAAS alias that you defined in a previous step.
 - c. Click **OK**, then **Save** your configuration.

5. Assign roles.

Roles must be assigned to authorize access to the bus and input and output bus destinations. These role assignments can be performed in the administrative console: **Security > Bus security > bus_name > Disabled > Users and groups in the bus connector role**.

You can also assign roles using either of the following `wsadmin` commands:

- `$AdminTask addUserToBusConnectorRole {-bus busName -user username}`
- `$AdminTask addGroupToBusConnectorRole {-bus busName -group groupname}`

Make the following role assignments:

- a. `JobSchedulerBus` Assign the `BusConnector` role to the following user IDs:
 - `com.ibm.ws.grid.ActivationSpec` to permit the job scheduler to access the bus.
 - Each identity used by `WSGrid` to authenticate client access to the input queue (see step 4). The `WSGrid` invoker then has permission to access the bus.
- b. Permit access to the `com.ibm.ws.grid.InputQueue` destination.

Permit access to this destination by assigning sender, receiver, and browser roles to the same user IDs. These IDs are the same IDs that you assigned the `BusConnector` role in the previous step.

You can permit access only through `wsadmin` commands:

- In `Jacl`:

```
$AdminTask addUserToDestinationRole {-type queue -bus JobSchedulerBus
  -destination com.ibm.ws.grid.InputQueue -role Sender -user userName}
```

or

```
$AdminTask addGroupToDestinationRole {-type queue -bus JobSchedulerBus
  -destination com.ibm.ws.grid.InputQueue -role Sender -group groupName}
```

- In `Jython`:

```
AdminTask.setInheritDefaultsForDestination('-bus WSS.JobScheduler.Bus -type queue
  -destination com.ibm.ws.grid.InputQueue -inherit false')
```

Repeat for receiver and browser roles.

- c. `com.ibm.ws.grid.OutputQueue`

Permit access to this destination by assigning the same roles to destination `com.ibm.ws.grid.OutputQueue` as were assigned for `com.ibm.ws.grid.InputQueue` in the previous step.

6. Authenticate client access to the input queue.

- a. Specify the user ID and password properties in the WSGrid input control properties file.

```
submitter-userid=username  
submitter-password=password
```

- b. Optional: Encode the password using the WebSphere PropFilePasswordEncoder utility.

Setting up the external scheduler interface using WebSphere MQ

You can install and configure the batch high performance external scheduler connector. This connector is the native WSGrid connector that is implemented in a native compiled language and that uses WebSphere MQ for communication.

About this task

The benefit of native WSGrid is twofold:

1. It makes more efficient use of z/OS system processors by preventing the need for Java virtual machine (JVM) startup processing on each use.
2. It uses the most robust messaging service available on z/OS to ensure reliable operation with a messaging service already known and used by most z/OS customers.

The authenticated user ID of the environment that starts WSGRID is propagated to the batch job scheduler. The resulting batch job runs by using that user ID. This user ID must also have sufficient WebSphere privileges to submit batch jobs, that is, lradmin or lrsubmitter roles. For example, if JCL job WSGRID1 is submitted to run under technical user ID TECH1, the resultant batch job also runs under user ID TECH1. User ID TECH1 must be permitted to get and put to and from the WebSphere MQ input and output queues used by WSGRID.

Procedure

1. Define WebSphere MQ queues.

Queue manager must be local. Two queues are required: one for input, one for output. You can name the queues according to your naming conventions. As an example, the name WASIQ is used for input queues and the name WASOQ is used for output queues. The queues must be set in shared mode.

2. Update the MQ_INSTALL_ROOT WebSphere variable.

- a. In the administrative console, click **Environment** > **WebSphere variables**.
- b. Select the node scope where the job scheduler runs.
- c. Select **MQ_INSTALL_ROOT**.
- d. For **Value**, put in the directory path to where WebSphere MQ is installed.
For example, **Value** can be /USR/1pp/mqm/V6R0M0.
- e. Click **Apply** and save the changes.

3. From the deployment manager, run the **installWSGridMQ.py** script with the following input parameters:

The **installWSGridMQ.py** script installs a system application, and then sets up the JMS connection factory, JMS input and output queues, and other necessary parameters.

```
wsadmin.sh -f -user <username> -password <userpassword> installWSGridMQ.py
```

```
-install  
  {-cluster <clusterName> | -node <nodeName> -server <server>}  
  
-remove  
  {-cluster <clusterName> | -node <nodeName> -server <server>}  
  
-qmgr  
  <queueManagerName>  
  
-inqueue  
  <inputQueueName>
```

-outqueue
<outputQueueName>

For example, for clusters:

```
wsadmin.sh -f installWSGridMQ.py -install -cluster <clusterName> -qmgr <queueManagerName>
-inqueue <inputQueueName> -outqueue <outputQueueName>
```

For example, for nodes:

```
wsadmin.sh -f installWSGridMQ.py -install -node <nodeName> -server <serverName>
-qmgr <queueManagerName> -inqueue <inputQueueName> -outqueue <outputQueueName>
```

4. Run **osgiCfgInit.sh|.bat -all** for each server whose MQ_INSTALL_ROOT WebSphere variable you updated in a previous step.

The **osgiCfgInit** command resets the class cache that the OSGi runtime environment uses.

5. Create the WSGRID load module:

- a. Locate the unpack script in the *app_server_root/bin* directory.

The **unpackWSGRID** script is a REXX script.

- b. Perform an unpack using the **unpackWSGrid** script. To display the command options, issue the **unpackWSGRID** script with no input: **unpackWSGRID <was_home> [<hlq>] [<work_dir>] [<batch>] [<debug>]**

<was_home>

Specifies the required WebSphere Application Server home directory.

<hlq> Specifies the optional high-level qualifier of output data sets default = <user id>.

<work_dir>

Specifies the optional working directory. The default is /tmp.

<batch>

Specifies the optional run mode for this script. Specify batch or interactive. The default is interactive.

<debug>

Specifies the optional debug mode. Specify debug or nodebug. The default is nodebug.

```
/u/USER26> unpackWSGRID /WebSphere/ND/AppServer
```

Sample output:

```
Unpack WSGRID with values:
WAS_HOME=/WebSphere/ND/AppServer
HLQ =USER26
WORK_DIR=/tmp
BATCH =INTERACTIVE
DEBUG =NODEBUG
Continue? (Y|N)
Y
User response: Y
Unzip /WebSphere/ND/AppServer/bin/cg.load.xmi.zip
extracted: cg.load.xmi
Move cg.load.xmi to /tmp
Delete old dataset 'USER26.CG.LOAD.XMI'
Allocate new dataset 'USER26.CG.LOAD.XMI'
Copy USS file /tmp/cg.load.xmi to dataset 'USER26.CG.LOAD.XMI'
Delete USS file /tmp/cg.load.xmi
Delete old dataset 'USER26.CG.LOAD'
Go to TSO and issue RECEIVE INDSN('USER26.CG.LOAD.XMI') to create
CG.LOAD
```

- c. Go to TSO, ISPF, option 6, and do a receive operation.

For example:

```
RECEIVE INDSN('USER26.CG.LOAD.XMI')
```

The following output is the result:

```
Dataset BBUILD.CG.LOAD from BBUILD on PLPSC
The incoming data set is a 'PROGRAM LIBRARY'
Enter restore parameters or 'DELETE' or 'END' +
```

Click **Enter** to end. Output similar to the following output is displayed.

```

IEB1135I IEBCOPY FMID HDZ11K0 SERVICE LEVEL UA4
07.00 z/OS 01.07.00 HBB7720 CPU 2097
IEB1035I USER26 WASDB2V8 WASDB2V8 17:12:15 MON
COPY INDD=((SYS00006,R)),OUTDD=SYS00005
IEB1013I COPYING FROM PDSU INDD=SYS00006 VOL=CPD
USER26.R0100122
IEB1014I
IGW01551I MEMBER WSGRID HAS BEEN LOADED
IGW01550I 1 OF 1 MEMBERS WERE LOADED
IEB147I END OF JOB - 0 WAS HIGHEST SEVERITY CODE
Restore successful to dataset 'USER26.CG.LOAD'
***

```

6. Restart the servers that you just configured. Also, restart the node agents.

Results

You have configured an external job scheduler interface.

What to do next

Submit a job from the external job scheduler interface to batch.

Requirements-based job scheduling

Batch provides both implicit and explicit job scheduling requirements, which match against endpoints based on an application name.

Implicit requirement matching

An implicit scheduling requirement determines which applications a job runs. In Java Platform, Enterprise Edition (Java EE) applications, the application name of all Java EE applications installed in the scheduling domain is known through the product configuration, because all Java EE applications are installed into this environment through product administrative interfaces.

For batch utility applications, this information cannot be assumed. By default the information is unavailable, since batch utility applications are installed outside the control of product administrative interfaces. For these applications, the batch administrator can optionally enable application-name based endpoint selection for particular nodes by setting the `grid.apps` node custom property.

Through this property, the administrator establishes configuration knowledge of the grid utility applications available on that node. When the `grid.apps` node custom property is set for a node, that node is an eligible endpoint for job dispatch if and only if all the batch utility applications named in a given xJCL are present in the `grid.apps` list. If the `grid.apps` node custom property is not set on a node, then application names are not considered during the endpoint selection process for that node. When there is a combination of nodes that either specify or do not specify the `grid.apps` node custom property, an order of precedence governs endpoint selection. The following code shows the order of precedence.

```

if any node published grid.apps then
  attempt to match job using app names
  if there is a match then
    if job has required-capabilities then
      candidate-nodes= $(apply required-capabilities to the set of nodes that have
matching apps)
    else
      candidate-nodes= $(nodes that have matching apps)
    endif
else (no match based on grid.apps)
  if job has required-capabilities then
    candidate-nodes= $(apply required-capabilities to all nodes)
  else
    candidate-nodes= $(all nodes)
  endif
endif

```



```
else (no nodes publish grid.apps)
  if job has required-capabilities then
    candidate-nodes= $(apply required-capabilities to all nodes)
  else
    candidate-nodes= $(all nodes)
  endif
endif
```

Java EE type applications always use application name matching during endpoint selection.

Explicit requirement matching

Explicit requirement matching enables a job to specify particular requirements that must be met by eligible endpoints. These requirements are specified in the batch job xJCL on the <required-capability> attribute. When required capabilities are specified, only endpoints that advertise matching capabilities are eligible to receive the job. The match expressions can specify any of the following node properties:

1. Node custom properties
2. Node metadata properties. For example, com.ibm.websphere.xdProductVersion
3. Node name and node host name

Requirements matching precedence

Implicit and explicit matching work together to determine eligible endpoints. For Java EE type jobs, the implicit requirement, application name, is treated as an additional required capability; it is logically appended with any explicitly specified requirements.

For batch utility jobs, an application name is a conditional requirement that applies only for batch utility nodes that advertise their installed applications through the grid.apps property. The batch utility jobs for which no application name match exists and that do not specify explicit requirements are eligible for dispatch to any batch utility node that does not advertise its applications.

Service policies for batch jobs

Batch support provides a rule-based service policy methodology and a completion time service policy goal type. It enhances workload management for batch because batch jobs and OLTP workloads can mix in the same product installation.

You must have batch installed to define service policies. See topics about dynamic operations.

Service policy classification is controlled by a set of rules defined to the job scheduler. These rules can be composed of Boolean expressions and can include the following operand types:

- Submitter identity
- Submitter group
- Job name
- Job class
- Application name
- Application type
- Platform
- Time

See “Batch job classification” on page 55 for details on the operand types.

Goals types

- Discretionary goals indicate work that does not have significant value. As a result, work of this type can see a degradation in performance when resources are constrained.
- Average response time goals are indicative of work with a higher priority than discretionary. The average response time goal is assigned a specific time goal.
- Percentile response time goals are another measure for work with a higher priority than discretionary. The percentile response goals are defined with specific criteria. The percentile response time target is the percentage of requests whose response time is T or less that should be P or more. A target has particular values for T and P.

Default classification rules and precedence

Batch support provides two default classification rules:

1. A rule that assigns any job of type Java Platform, Enterprise Edition (Java EE) to the transaction class defined by default IIO work class of the named Java EE application.
2. A rule that assigns any job to the default transaction class, DEFAULT_TC

Both default rules can be edited and deleted. The order of the rules can be modified, and user-defined classification can be added. The job scheduler evaluates the list of classification rules in order and assigns the transaction class specified by the first matching rule. Only one classification rule set for a cell is supported. A default configurable transaction class, named DEFAULT_TC by default, is associated with this set. If none of the classifications rules match a job, then the default transaction class is applied to that job. Graphical user interface (GUI) support for choosing a transaction class from a list while building a rule is only present when batch support is installed. When batch alone exists, there is a text field where a transaction class name is specified.

Batch job classification

The classification rules are composed of Boolean expressions composed of the operands listed in this topic.

Classification rules are saved in a configuration file named `gridclassrules.xml` under the profile config directory. The rules are ordered based on the priority element.

The batch runtime implementation iterates through these rules until it finds a matching rule. When a match is found, the transaction class corresponding to that rule is applied to that job.

The following operands comprise the Boolean expressions in job classification rules:

submitterid

The submitter of a given job is obtained from `com.ibm.ws.longrun.Job.getUser()` API. It must match the value of this operand for the rule to match.

submittergroup

The submitter group for a job is a list of strings obtained by iterating through the set of credentials for that job submitter. If the value of this operand exists in the list, then the rule is matched and the corresponding transaction class is assigned to the job.

jobname

The name attribute of the job element in xJCL specifies the job name. It must match the value of this operand for the rule to match.

jobclass

The optional class attribute of the job element in xJCL specifies the job class. It must match the value of this operand for the rule to match.

appname

The optional default-application-name attribute of the job element in xJCL specifies this name. If this attribute is not specified, then appname defaults to job name. The application name of a given job must match the value of this operand for the rule to match.

apptype

Valid values for this operand are GridUtility and J2EE. If the application type of the job matches the value of this operand, then this rule is matched. The corresponding tx class of the rule is assigned to the job.

platform

Valid values for this operand are zos, distributed, and mixed. If the platform composition of the eligible endpoints for a given job matches the value of this operand, then the rule matches.

time

Use the time operand to define the date and time of day that a given request must be honored. Two optional fields are StartTime and EndTime. If a request is received outside of the defined window, the request is not processed. The format for both fields is *dayOfWeek/day/month/year:hour:min:sec*, for example, Sat/01/Jan/2011::08:00:00.

dayOfWeek

Specifies one of the days of the week: Sun for Sunday, Mon for Monday, Tue for Tuesday, Wed for Wednesday, Thu for Thursday, Fri for Friday, and Sat for Saturday.

day

Specifies the day of the month.

month

Specifies one of the 12 months: Jan for January, Feb for February, Mar for March, Apr for April, May for May, Jun for June, Jul for July, Aug for August, Sep for September, Oct for October, Nov for November, Dec for December.

year

Specifies the four-digit year.

hour

Specifies the two-digit hour of the 24 hour clock.

min

Specifies the two-digit value for minutes.

sec

Specifies the two-digit value for seconds.

Sample classification rules

```
<matchRules xmi:id="MatchRule_1159377240783" matchAction=SimpleCI_TC" matchExpression="apptype='j2ee'"
priority="1"/>
```

```
<matchRules xmi:id="MatchRule_1159377240783" matchAction="CompletionTime_TC"
matchExpression="appname='MandlebrotCI" priority="2"/>
```

```
<matchRules xmi:id="MatchRule_1159377240783" matchAction="{default_iiop_transaction_class}"
matchExpression="submitterid='admin'" priority="3"/>
```

Default classification rules and precedence

The default classification rule assigns any job to the default transaction class, DEFAULT_TC.

The default rule can be edited and deleted. User-defined classification can be added. The job scheduler evaluates the list of classification rules in order and assigns the transaction class specified by the first matching rule. Only one classification rule set per cell is supported. A default configurable transaction class, named DEFAULT_TC by default, is associated with this set. If none of the classifications rules match a job, then the default transaction class is applied to that job. GUI support for choosing a transaction class from a list while building a rule is available only when Intelligent Management is installed. When only batch exists in your environment, there is a text field where a transaction class name is specified.

Note: Assign transaction classes to batch work on the Job scheduler Classification rules administrative console page, not on the **Resources > Asynchronous beans > Work managers > BatchWorkManager** page.

Job usage data for charge-back accounting support

The product provides charge-back accounting information for batch jobs for all operating systems. You can use charge-back accounting to determine the computing costs of batch job execution for work that has been performed by various users and groups.

The job scheduler records usage data for charge-back accounting when enabled through the administrative console. This function is available for all operating systems and can be enabled and disabled through configuration settings. Job usage information is not enabled by default.

Accounting information for each job includes:

- Job identity: The job identifier associated with the job, which is returned by the job scheduler when the job is submitted
- Submitter identity: The identity (if any) of the submitter of the job
- CPU used: An integer that is the number of units of CPU used, where one unit is 10^{-6} second.
- Job state: The state of the job. That is, running or ended.
- Node name and server name: The node and server names where the job ran
- Job start time: The time the job began to run
- Last update time: The time of the last job usage update
- Accounting string: The job accounting information associated with the job

The DDL statements for the job scheduler JOBUSAGE table are defined in the CreateLRSCHEdTables*.ddl files of the `app_server_root/util/Batch` directory. An example schema definition is:

```
CREATE TABLE "LRSSHEMA"."JOBUSAGE" (  
  "JOBID" VARCHAR(250) NOT NULL ,  
  "SUBMITTER" VARCHAR(256),  
  "CPUCONSUMEDSO FAR" BIGINT NOT NULL,  
  "JOBSTATE" VARCHAR(32) NOT NULL,  
  "SERVER" VARCHAR(250) NOT NULL,  
  "NODE" VARCHAR(250) NOT NULL,  
  "STARTTIME" VARCHAR(64) NOT NULL,  
  "LASTUPDATE" VARCHAR(64) NOT NULL,  
  "ACCTING" CHAR(64) ) IN "USERSPACE1" ;
```

The table data can be accessed with an SQL query, for example:

```
select * from LRSSHEMA.JOBUSAGE where JOBID='PostingSampleEar:99'
```

Programmatic access to the scheduler job usage table data must specify an isolation level of read uncommitted, to impede active job execution or the recording of accounting data. The STARTTIME and LASTUPDATE represent the return value of System.currentTimeMillis(). See ++ in the following information for more detail.

z/OS

Job usage SMF record layout

The purpose of the JobUsage SMF record is to record information about a batch container job inside a WebSphere Application Server for z/OS transaction server. The SMF type 120 record, subtype 20, is the job usage record. There is one job usage section per record as shown in the following table:

Table 21. Job usage SMF record layout. The table includes the offset in decimal, the offset in hexadecimal, the name, the length, the format, and a description.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120XVL	2	binary	Length of the JobUsage section
2	2	SM120XJL	1	binary	Length of the job identifier field; maximum is 250
3	3	SM120XJ	250	EBCDIC	Job identifier
254	FE	SM120XT	32	EBCDIC	Job submitter
286	11E	SM120XSL	1	binary	Length of the job state field; maximum is 32
287	11F	SM120XS	31	EBCDIC	Job state (final). Can be one of: ended, execution failed, or restartable
319	13F	SM120XNL	1	binary	Length of the server name field; maximum is 250
320	140	SM120XN	250	EBCDIC	Server name.
570	23A	SM120XOL	1	binary	Length of the node name field; maximum is 250
571	23B	SM120XO	250	EBCDIC	Node name
821	335	SM120XAL	1	binary	Length of the accounting information field; maximum is 64
822	336	SM120XA	64	EBCDIC	Accounting information.
886	376	SM120XBL	1	binary	Length of the job start time field; maximum is 64
887	377	SM120XB	64	EBCDIC	Job start time **
951	3B7	SM120XLL	1	binary	Length of the last update time field; maximum is 64
952	3B8	SM120XL	64	EBCDIC	Last update time**
1016	3F8	SM120XPL	1	binary	Length of the CPU consumed field; maximum is 64
1017	3F9	SM120XP	64	EBCDIC	Total CPU consumed in microseconds ^ ^
1081	439	SM120XZ	64	EBCDIC	CPU time in microseconds on general purpose processors ^ ^

** The offsets of the fields which follow are based upon a full field. The field contains data that is the maximum field length. The actual offset in the record is the start of the field, plus the length of the field, plus one.

** Represented as a character string that is the number of milliseconds since January 1 1970, 00:00:00 GMT.

^ ^ Represented as a character string that is the number of units of CPU used, where one unit is 10⁻⁶ seconds. 10⁻⁶ means 10 raised to the -6th power; that is, .000001.

z/OS

Job usage with SMF 120 subtype 9 records

Job usage information can be recorded with either SMF 120 subtype 20 records or SMF 120 subtype 9 records. SMF 120 subtype 20 records are described in this topic in the section on job usage SMF record layout.

SMF 120 subtype 9 records contain many of the metrics included in the SMF 120 subtype 20 records. Additionally, the batch container adds the job ID, the submitter ID, and the job accounting string to the user data section of the SMF 120 subtype 9 record.

Note: SMF 120 subtype 9 support for batch jobs requires that SMF 120 subtype 9 recording for asynchronous beans is enabled on the endpoint server. SMF 120 subtype 9 support for asynchronous beans is available on WebSphere Application Server Version 8.0.0.1 or later. Earlier versions are not supported. If you specify `RECORD_SMF_SUBTYPES=9` on an earlier version, the job scheduler issues a message. The message indicates that SMF 120 subtype 9 records are not supported on earlier versions of WebSphere Application Server. The job scheduler reverts to SMF 120 subtype 20 records.

Note: If you specify `RECORD_SMF_SUBTYPES=9` without also enabling SMF 120 subtype 9 recording for asynchronous beans in the endpoint server, the endpoint server issues a message. The message indicates that SMF 120 subtype 9 recording for asynchronous beans is not enabled. No SMF120 subtype 9 job usage records are collected.

The batch container user data type is 101 decimal or x65 hexadecimal. The data has a fixed length of 352 decimal or x160 hexadecimal. All of the fields have fixed lengths. The format of the user data is described in the following table.

Table 22. Format of the user data. The table lists the decimal offset, the hexadecimal offset, the length, the format, and the description of the user data.

Decimal Offset	Hexadecimal offset	Length	Format	Description
0	0	1	Binary	Length of job ID
1	1	255	EBCDIC	Job ID
256	100	32	EBCDIC	Submitter ID
288	120	1	Binary	Length of accounting information
289	121	63	EBCDIC	Accounting information

z/OS

Formatting batch SMF records using the SMF Browser

Batch SMF 120 subtype 20 and subtype 9 records can be formatted using the batch add-on to the SMF Browser for WebSphere Application Server for z/OS.

Follow the instructions packaged within the add-on and within the SMF Browser utility to invoke the browser.

Integrating batch features in z/OS operating systems

Use these tasks to integrate batch features into the z/OS environment.

Procedure

- Manage multi-user Workload Manager (WLM) environments. See “Managing multi-user WLM environments” on page 61 for more information.
- Manage worker threads. See “Managing worker threads” on page 61 for more information.

z/OS workload management and service policies

If users who work in a multi-user address space are assigned to the same service policy, normal WLM behavior is assured. All user work in such an address space receives equal treatment. In an operating system dispatcher queue priority, the z/OS workload management (WLM) manages running work at the address space level.

Since traditional z/OS workloads were either one job or one online user at a time per address space, the service policy of the address space is considered when determining dispatcher priority. When multi-user address spaces, such as CICS® or WebSphere Application Server were introduced, the fundamental WLM/dispatcher model was not changed, but rather accommodated.

However, in a multi-user address space where the user work is assigned to mixed service policies, only the most aggressive service policy is considered for purposes of managing dispatcher priority. The service policy with the most stringent performance goal and highest importance governs the dispatch priority of such an address space.

Running mixed service policy in a single address space has a negative consequence. The work of lesser importance in such an address space receives the same dispatcher prioritization as the most important work in the address. This situation occurs because the most aggressive service policy governs that prioritization. An attempt by the user to achieve service policy differentiated resource management is undermined.

Transaction class propagation on z/OS operating systems

Service policies contain one or more transaction class definitions. The service policy creates the goal, while the job transaction class is used to connect the job to that goal. When working with z/OS resident applications, the goal defined in the service policy is only used for monitoring and reporting rather than active workload management. The transaction class also serves the purpose of providing the TCLASS value that is propagated to the request and used by Workload Manager for z/OS (WLM).

Transaction classes

Transaction classes are a subcontainer of the service policy for work being classified into the service policy that can be used for finer-grained monitoring. The relationship between service policies and transaction classes is one to many: A single service policy can have multiple transaction class definitions, but each transaction class belongs to exactly one service policy. Every service policy has a default transaction class, which in most scenarios is sufficient. Additional transaction classes are created when finer-grained monitoring is necessary for the environment. Each transaction class name must be unique within the cell.

In the batch environment each job is assigned to a job class. A job class establishes a policy for resource consumption by a set of batch jobs. If a job does not specify a job class, a default one is provided.

Service policy classification in the batch environment is controlled by a set of rules defined to the job scheduler: A rule that assigns any job to the default transaction class DEFAULT_TC.

The job scheduler evaluates the list of classification rules in order and assigns the transaction class specified by the first matching rule. Only one classification rule set per cell is supported. A default configurable transaction class, DEFAULT_TC by default, is associated with this set. If none of the classification rules match a job, then the default transaction class is applied to that job. When only a batch environment exists there is a field where you specify a transaction class name.

When a job dispatch request reaches the control region, the TCLASS is extracted from the HTTP request header and used to associate the request with a WLM for z/OS service class. An enclave is created having the indicated service class and it is dispatched using WLM to a servant region where the job is run. Queuing and prioritization to achieve service class goals is done by WLM for z/OS.

Managing multi-user WLM environments

Use the WebSphere Application Server workload management (WLM) to control the performance of and the number of application server regions in z/OS.

Before you begin

You must define one or more server instances on one or more systems within the WebSphere node. Each server instance consists of a controller region and one or more servant regions. You must start controller regions as MVS started tasks, while servant regions are started automatically by WLM on an as-needed basis.

About this task

You can deploy applications within a WebSphere generic server. WebSphere servers are configured, by default, to allow only one servant region. The application server administrator must use the administrative console to allow the generation of multiple servant regions.

Procedure

- Configure the application servants setting. Configure the application server minimum/maximum servants setting as follows:
Min servants <= number of possible service policies <= max servants
- To change the minimum and maximum number of servant instances using the administrative console, select **Servers > Server Types > WebSphere application servers > server_name**. Click **Java and Process Management > Server instance**. Check the box **Multiple Instances Enabled** and type the minimum and maximum number of servant instances.

Managing worker threads

Use this topic to support Object Request Broker (ORB) service advanced settings. The workload profile specifies the server workload profile, which can be ISOLATE, IOBOUND, CPUBOUND, or LONGWAIT.

About this task

Not only does workload management (WLM) dispatch work to servants according to service policy, but it also does so only if it has available worker threads. WLM worker threads are regular threads that specifically register with WLM as work receivers. In the WebSphere Application Server for z/OS implementation, this pool of threads is static. The pool in an address space does not grow or contract. The number of worker threads governs the maximum number of concurrent requests that WLM accepts in a servant. However, this situation applies only to HTTP, IIOP, and Java Message Service (JMS) driven requests. This thread pool does not handle asynchronous beans. The number of threads allocated to this pool is governed by an external object known as the ORB Workload profile.

Procedure

1. To configure the workload profile in the administrative console, click **Servers > Server Types > WebSphere application servers > server_name > Container services > ORB service > z/OS additional settings**.

ISOLATE

Number of threads is 1. Specifies that the servants are restricted to a single application thread. Use ISOLATE to ensure that concurrently dispatched applications do not run in the same servant. Two requests processed in the same servant can cause one request to corrupt another.

IOBOUND

Default - Number of threads is 3 * Number of processors. Specifies more threads in applications that perform I/O-intensive processing on the z/OS operating system. The

calculation of the thread number is based on the number of processors. IOBOUND is used by most applications that have a balance of processor intensive and remote operation calls. A batch job is an example that uses the IOBOUND profile.

CPUBOUND

Number of threads is the number of processors. Specifies that the application performs processor-intensive operations on the z/OS operating system, and therefore, would not benefit from more threads than the number of processors. The calculation of the thread number is based on the number of processors. Use the CPUBOUND profile setting in processor intensive applications, like compute-intensive (CI) jobs, XML parsing, and XML document construction, where the vast majority of the application response time is spent using the processor.

LONGWAIT

Number of threads is 40. Specifies more threads than IOBOUND for application processing. LONGWAIT spends most of its time waiting for network or remote operations to complete. Use this setting when the application makes frequent calls to another application system, like Customer Information Control System (CICS) screen scraper applications, but does not do much of its own processing.

2. To change the minimum and maximum number of servant instances using the administrative console, select **Servers > Server Types > WebSphere application servers > server_name**. Under **Server Infrastructure**, click **Java and Process Management > Server Instance**. Check the box **Multiple Instances Enabled**, and type the minimum and maximum number of servant instances.

Min servants <= number of possible service policies <= max servants

Results

Number of processors is the number of processors online when the controller starts. You can look at message BBOO0234I in the controller job log to check the number of worker threads.

Enabling job usage information

The job scheduler records usage data for charge-back accounting when enabled through the administrative console. This function is available for all operating systems and can be enabled and disabled through configuration settings. Job usage information is not enabled by default.

Before you begin

Configure the job scheduler.

z/OS Read the topic about the SMF record type 120. Also, see the *z/OS MVS System Management Facility (SMF)* manual, order number SA22-7630, for additional information about using SMF records.

Procedure

1. In the administrative console, click **System administration > Job scheduler**.
The Job scheduler page provides configuration settings for job usage.
2. Select **Record usage data in scheduler database** to record job usage information to the job usage database table.
3. **z/OS** Select to record job usage information to the System Management Facility (SMF) data set. SMF recording is activated and deactivated along with the activation and deactivation of SMF recording for the Java Platform, Enterprise Edition (Java EE) container.
Job usage data can be recorded with either SMF 120 subtype 9 records or SMF 120 subtype 20 records. Use the RECORD_SMF_SUBTYPES job scheduler custom property to indicate the preferred subtype.
 - a. If you are collecting SMF 120 subtype 9 records, set properties that enable SMF request activity.

You can use the administrative console or MVS Modify commands to set properties that enable SMF request activity.

- Using the administrative console:
 - 1) Click **Servers > Server Types > WebSphere application servers > *server_name* > Java and Process Management > Process definition > Environment entries > New.**
 - 2) Create two environment entry properties.
 - Specify a name of `server_SMF_request_activity_enabled` and value of 1 (or true) for one property.
 - Specify a name of `server_SMF_request_activity_async` and value of 1 (or true) for the other property.
- Using the MVS Modify commands, set the properties to on:
 - `f server_name,smf,request,on`
 - `f server_name,smf,request,async,on`
- b. On the Job scheduler page of the administrative console, select **Record usage data in SMF (z/OS only)**.
- c. Click **Custom properties** to go to the Job scheduler custom property page. Set the `RECORD_SMF_SUBTYPES` custom property to 9 for subtype 9 records or to 20 for subtype 20 records.

Rolling out batch application editions

Batch applications are Java Platform, Enterprise Edition (Java EE) applications that conform to one of the batch programming models. By rolling out an edition, you replace an active edition with a new edition.

Before you begin

You must have an application edition that is installed and started, and have configurator or administrator administrative privileges to perform this task.

Note: Application edition rollout fails when two user IDs on two administrative consoles attempt a parallel application edition rollout.

About this task

The new edition might be a simple modification to the application, such as a bug fix, or a more substantial change. As long as the new edition is compatible with earlier versions, you can perform a rollout to replace the active edition without affecting existing clients. To roll out a new edition, you must first install the application edition with the new edition information.

Procedure

1. Install the new edition. Specify your new edition information. Use the following example to specify the new edition.
 - a. Type 2.0 in the **Application edition** field.
 - b. Type Second edition in the **Application description** field.
 - c. Select the same deployment targets that are used for the current edition.
2. Save and synchronize your nodes.
3. Start the rollout. Click **OK**. This action starts an interruption-free replacement of the older edition with your new edition.

Results

For an edition that is not in validation mode, the new edition replaces the current edition after the rollout completes. An edition that is in validation rolls out on the original deployment target and the cloned environment is deleted. If the jobs of the batch rollout application are running on the quiesced endpoint, the job scheduler will cancel the jobs after the drainage time.

What to do next

To validate the results, select **Applications > Edition Control Center > *application_name***. Your new edition is the active edition on the deployment target. The new edition automatically starts, because it replaces a running edition.

When an application edition in validation mode is rolled out, the binding names must be changed back to the original values. For example: `/clusters/cluster1-validation/jdbc/CustomerData` must be changed back to `/clusters/cluster1/jdbc/CustomerData`.

Job scheduler custom properties

Custom properties modify the job scheduler configuration. You can use these settings to tune the job scheduler behavior beyond the settings that are in the administrative console.

You can use the custom properties page to define the following job scheduler custom properties:

- “MaxConcurrentDispatchers”
- “UseHTTPSConnection”
- “RECORD_SMF_SUBTYPES” on page 65
- “JOB_SECURITY_POLICY” on page 65
- “JOB_SECURITY_DEFAULT_GROUP” on page 65
- “JOB_SECURITY_ADMIN_GROUP” on page 66
- “UseAPCEndpointSelection” on page 66
- “WXDBulletinBoardProviderOption” on page 66

MaxConcurrentDispatchers

Define this custom property if jobs are being dispatched slowly when large numbers of jobs are submitted. By default, MaxConcurrentDispatchers is set to 100. MaxConcurrentDispatchers is an optional custom property.

Table 23. MaxConcurrentDispatchers custom property values. The table includes the scope, valid values, and default for the custom property.

Scope	Valid values	Default
Job scheduler	Integer value greater than 0	100

UseHTTPSConnection

Define this custom property if you want to enable HTTP SSL connections between the job scheduler and the common batch container. By default, HTTP SSL connections are disabled. UseHTTPSConnection is an optional custom property.

Table 24. UseHTTPSConnection custom property values. The table includes the scope, valid values, and default for the custom property.

Scope	Valid values	Default
Job scheduler	<ul style="list-style-type: none"> • true Enables HTTP SSL connections • false Disables HTTP SSL connections 	false (disabled)

RECORD_SMF_SUBTYPES

Define this property to indicate which SMF 120 record subtype you want to use to record job usage data. By default, SMF 120 subtype 20 records are used. RECORD_SMF_SUBTYPES is an optional custom property.

Table 25. RECORD_SMF_SUBTYPES custom property values. The table includes the scope, valid values, and default for the custom property.

Scope	Valid values	Default
Job scheduler cell	<ul style="list-style-type: none"> • 20 Use SMF120 subtype 20 records. • 9 Use SMF120 subtype 9 records. Note: SMF 120 subtype 9 support for batch jobs requires that SMF 120 subtype 9 recording for asynchronous beans is enabled on the endpoint server. SMF 120 subtype 9 support for asynchronous beans is available on WebSphere Application Server Version 8.0.0.1 or later. Earlier versions are not supported. If you specify RECORD_SMF_SUBTYPES=9 on an earlier version, the job scheduler issues a message. The message indicates that SMF 120 subtype 9 records are not supported on earlier versions of WebSphere Application Server. The job scheduler reverts to SMF 120 subtype 20 records. Note: If you specify RECORD_SMF_SUBTYPES=9 without also enabling SMF 120 subtype 9 recording for asynchronous beans in the endpoint server, the endpoint server issues a message. The message indicates that SMF 120 subtype 9 recording for asynchronous beans is not enabled. No SMF120 subtype 9 job usage records are collected. • ALL Use both SMF120 subtype 20 and SMF120 subtype 9 records. 	20

JOB_SECURITY_POLICY

Define this property to indicate whether administrative roles, groups, or a combination of the two can act on a set of jobs.

Table 26. JOB_SECURITY_POLICY custom property values. The table includes the scope, valid values, and default for the custom property.

Scope	Valid values	Default
Job scheduler	<ul style="list-style-type: none"> • ROLE Specifies that the batch domain applies role-based job security. This behavior is consistent with version 6.1.1 of batch. • GROUP Specifies that the batch domain uses only group-affiliation as the basis for job security. • GROUPROLE This setting specifies that the batch domain uses group and role as the basis for job security. 	ROLE

JOB_SECURITY_DEFAULT_GROUP

Define this custom property so that you can assign a user group to a job. You can use this property if either group or group and role security is active.

You can also assign a group to a job using the group attribute on the job definition in the xJCL. The group attribute takes precedence over the value on the JOB_SECURITY_DEFAULT_GROUP custom property. If you do not specify the group attribute on the job definition or this custom property, the job scheduler assigns the default group name.

Table 27. JOB_SECURITY_DEFAULT_GROUP custom property values. The table includes the scope, valid values, and default for the custom property.

Scope	Valid values	Default
Job scheduler	A user-defined user group name	JSYSDFLT

JOB_SECURITY_ADMIN_GROUP

Define this custom property so that you can assign an administrative group to each job.

Table 28. JOB_SECURITY_ADMIN_GROUP custom property values. The table includes the scope, valid values, and default for the custom property.

Scope	Valid values	Default
Job scheduler	A user defined administrative group name	JSYSADMN

UseAPCEndpointSelection

Define this property if you want to use the application placement controller for job placement. You cannot use the application placement controller for job placement on the z/OS platform. Setting this property allows the application placement controller to select the endpoint. Otherwise, the grid application placement does the selection.

Table 29. WXDBulletinBoardProviderOption custom property values. The table includes the scope, valid values, and default for the custom property.

Scope	Valid values	Default
Job scheduler	true	false

WXDBulletinBoardProviderOption

Define this property if you want to use the application placement controller for job placement. You cannot use the application placement controller for job placement on the z/OS platform. Setting this custom property with a value of HAMBB overlays the base bulletin board for Intelligent Management with the high availability manager bulletin board. Batch and Intelligent Management can then use the same bulletin board to talk to each other.

Table 30. WXDBulletinBoardProviderOption custom property values. The table includes the scope, valid values, and default for the custom property.

Scope	Valid values	Default
Job scheduler cell	HAMBB	None

Port number settings for batch

Identify the default port numbers used in the various configuration processes to avoid port conflicts.

Open certain ports in any firewalls that are running between the deployment manager and node agent server processes to avoid conflicts with other assigned ports when configuring resources or assigning port numbers to other applications. Additionally, when you configure a firewall, enable access to specific port numbers.

If you modify the ports, or if you want to confirm the assigned port, check the port assignments in the `app_install_root/config/cells/cell_name/nodes/node_name/serverindex.xml` file. If more than one node exists, you must check the port assignment for each node.

The following table is a list of port assignments that the node agent server uses by default. When you federate an application server node into a deployment manager cell, the deployment manager instantiates the node agent server process on the application server node. The node agent server uses these port assignments by default. When multiple processes share a port on the same node, the configuration uses the next number in the sequence for the additional processes. For example, if three processes are running, the `BOOTSTRAP_ADDRESS` port for each process is 2809, 2810, and 2811.

For a complete list of default port definitions, see the topic on port settings. For information about security ports, see the topic on inbound ports.

Table 31. Default port definitions for the node agent server process. The table lists the port name, then a description and default value for the port.

Port Name	Description	Default Value (increments for multiple processes)
<code>BOOTSTRAP_ADDRESS</code>	The TCP/IP port on which the name service listens. This port is also the RMI connector port. Specify this port with the administrative console or with the <code>chgwassvr</code> script.	2809
<code>ORB_LISTENER_ADDRESS</code>	The TCP/IP port on which the application server Object Request Broker (ORB) listens for requests. The location service daemon for the node also listens on this port. Specify this port with the administrative console or with the <code>chgwassvr</code> script.	9100
<code>SAS_SSL_SERVERAUTH_LISTENER_ADDRESS</code>	The TCP/IP port on which the Secure Association Services (SAS) listen for inbound authentication requests. Specify this port with the administrative console or with the <code>chgwassvr</code> script.	9901
<code>CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS</code>	The TCP/IP port on which the Common Secure Interoperability Version 2 (CSIV2) Service listens for inbound client authentication requests. Specify this port with the administrative console or with the <code>chgwassvr</code> script.	9202
<code>CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS</code>	The TCP/IP port on which the Common Secure Interoperability Version 2 (CSIV2) Service listens for inbound server authentication requests. Specify this port with the administrative console or with the <code>chgwassvr</code> script.	9201
<code>NODE_DISCOVERY_ADDRESS</code>	The TCP/IP port on which the node discovery service for the node agent listens. Specify this port with the administrative console or with the <code>chgwassvr</code> script.	7272
<code>NODE_MULTICAST_DISCOVERY_ADDRESS</code>	The TCP/IP port for the multicast discovery service on which the node agent listens. Specify this port with the administrative console or with the <code>chgwassvr</code> script.	5000
<code>SOAP_CONNECTOR_ADDRESS</code>	This port is required by every WebSphere process to enable SOAP connectivity for JMX calls when using the <code>wsadmin</code> script.	8879
<code>OVERLAY_UDP_LISTENER_ADDRESS</code>	Used for peer-to-peer (P2P) communication. The ODC (On Demand Configuration) and asynchronous PMI components use P2P as their transport. This port is required by every WebSphere Extended Deployment process.	11001
<code>OVERLAY_TCP_LISTENER_ADDRESS</code>	Used for P2P communication. The ODC (On Demand Configuration) and asynchronous PMI components use P2P as their transport. This port is required by every WebSphere Extended Deployment process.	11002
<code>XD_AGENT_PORT</code>	The deployment manager, the node agents, and the middleware agents each have one <code>XD_AGENT_PORT</code> . Unlike the <code>OVERLAY</code> ports, the application servers are not configured with <code>XD_AGENT_PORT</code> ports. Used to enable communication between the deployment manager, the node agents, and the middleware agents. The ODR uses this port to collect information from other servers, including node agents. Ensure that this port is available to all servers that the Intelligent Management ODR is managing.	7061
<code>DRS_CLIENT_ADDRESS</code>	Deprecation: This port is deprecated and is no longer used in the current version of the product.	7873

During the `addNode` command operation, the filetransfer application uses port 9090 by default. The filetransfer application uses the same HTTP transport port that is used by the administrative console. If security is enabled, the default secured port 9043 must be opened in the firewall. If you modify the ports, or if you want to confirm the assigned port, check the port assignments in the `app_install_root/config/cells/cell_name/nodes/node_name/servers/dmgr/server.xml` file.

Table 32. Default port definitions for the `filetransfer` application. The tables includes the port number and its default value.

Port Name	Default Value
Default <code>filetransfer</code> application Port	9090
Secured - Default <code>filetransfer</code> application port	9043

When you federate an application server node with the embedded messaging server feature into a deployment manager cell, the deployment manager instantiates a Java Message Service (JMS) server process, `jmsserver`, on the application server node. The following table lists the port assignments that the JMS server uses by default:

Table 33. Default port definitions for the JMS server. The tables includes the port number and its default value.

Port Name	Default Value
<code>JMSSERVER_DIRECT_ADDRESS</code>	5559
<code>JMSSERVER_QUEUED_ADDRESS</code>	5558
<code>SOAP_CONNECTOR_ADDRESS</code>	8879
<code>JMSSERVER SECURITY PORT</code>	5557

Batch administrator examples

Batch administrator examples are examples of code snippets, command syntax, and configuration values that are relevant to performing administrative and deployment tasks in the batch environment.

The samples in this section consist of xJCL samples and XML schemas for batch jobs, native jobs, and compute-intensive jobs.

xJCL sample for a batch job

The following sample illustrates a batch job, which demonstrates that you can invoke existing session beans from within job steps.

```
<job name="PostingsSampleEar" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <jndi-name>ejb/com/ibm/websphere/samples/PostingsJob</jndi-name>

    <step-scheduling-criteria>
    <scheduling-mode>sequential</scheduling-mode>
    </step-scheduling-criteria>

    <checkpoint-algorithm name="{checkpoint}">
    <classname>com.ibm.wsspi.batch.checkpointalgorithms.{checkpoint}</classname>
    <props>
    <prop name="interval" value="{checkpointInterval}" />
    </props>
    </checkpoint-algorithm>

    <results-algorithms>
    <results-algorithm name="jobsum">
    <classname>com.ibm.wsspi.batch.resultsalgorithms.jobsum</classname>
    </results-algorithm>
    </results-algorithms>

    <substitution-props>
    <prop name="wsbatch.count" value="5" />
    <prop name="checkpoint" value="timebased" />
    <prop name="checkpointInterval" value="15" />
    <prop name="postingsDataStream" value="{was.install.root}{file.separator}temp{file.separator}postings" />
    </substitution-props>
</job>
```

```

<job-step name="Step1">

    <jndi-name>ejb/DataCreationBean</jndi-name>

    <!-- apply checkpoint policy to step1 -->
    <checkpoint-algorithm-ref name="{checkpoint}" />

    <results-ref name="jobsum"/>

</batch-data-streams>
<bds>

    <logical-name>myoutput</logical-name>

    <impl-class>com.ibm.websphere.samples.PostingOutputStream</impl-class>

    <props>
        <prop name="FILENAME" value="{postingsDataStream}" />
    </props>
</bds>
</batch-data-streams>

    <props>
        <prop name="wsbatch.count" value="{wsbatch.count}" />
    </props>
</job-step>

<job-step name="Step2">

    <step-scheduling condition="OR">
        <returncode-expression step="Step1" operator="eq" value="0" />
        <returncode-expression step="Step1" operator="eq" value="4" />
    </step-scheduling>

    <jndi-name>ejb/PostingAccountData</jndi-name>
    <checkpoint-algorithm-ref name="{checkpoint}" />
    <results-ref name="jobsum"/>

</batch-data-streams>
<bds>

    <logical-name>myinput</logical-name>
    <impl-class>com.ibm.websphere.samples.PostingStream</impl-class>

    <props>
        <prop name="FILENAME" value="{postingsDataStream}" />
    </props>

</bds>
</batch-data-streams>
</job-step>

    <job-step name="Step3">
<step-scheduling>
    <returncode-expression step="Step2" operator="eq" value="4" />
</step-scheduling>

    <jndi-name>ejb/OverdraftAccountPosting</jndi-name>
    <checkpoint-algorithm-ref name="{checkpoint}" />
    <results-ref name="jobsum" />

```

```

    <batch-data-streams>
      <bds>
        <logical-name>dbread</logical-name>
        <impl-class>com.ibm.websphere.samples.OverdraftInputStream</impl-class>
      </bds>
    </batch-data-streams>
  </job-step>
</job>

```

XML schema for a batch job

The following example shows the XML schema for a batch job:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="classname" type="xsd:string" />
  <xsd:element name="impl-class" type="xsd:string" />
  <xsd:element name="jndi-name" type="xsd:string" />
  <xsd:element name="logical-name" type="xsd:string" />

  <xsd:element name="scheduling-mode">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="sequential"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>

  <xsd:element name="required" >
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="[YNyn]"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>

  <xsd:element name="batch-data-streams">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" ref="bds" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="job-scheduling-criteria">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" ref="required-capability" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="bds">
    <xsd:complexType>
      <xsd:all>
        <xsd:element ref="logical-name" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="impl-class" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="props" minOccurs="0" maxOccurs="1"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="checkpoint-algorithm">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="classname" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="props" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required" />
    </xsd:complexType>
  </xsd:element>

```



```

    </xsd:complexType>
</xsd:element>

<xsd:element name="checkpoint-algorithm-ref">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:string" use="required" />
  </xsd:complexType>
</xsd:element>

<xsd:element name="required-capability">
  <xsd:complexType>
    <xsd:attribute name="expression" type="xsd:string" use="required" />
  </xsd:complexType>
</xsd:element>

<xsd:element name="results-algorithm">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="classname" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="props" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="required" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
  </xsd:complexType>
</xsd:element>

<xsd:element name="results-algorithms">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="1" ref="results-algorithm" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="results-ref">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:string" use="required" />
  </xsd:complexType>
</xsd:element>

<xsd:element name="substitution-props">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="prop" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="job">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="jndi-name" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="job-scheduling-criteria" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="step-scheduling-criteria" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="checkpoint-algorithm" maxOccurs="unbounded" minOccurs="1"/>
      <xsd:element ref="results-algorithms" maxOccurs="1" minOccurs="0"/>
      <xsd:element ref="substitution-props" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="job-step" maxOccurs="unbounded" minOccurs="1" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="class" type="xsd:string" use="optional" />
    <xsd:attribute name="accounting" type="xsd:string" use="optional" />
    <xsd:attribute name="default-application-name" type="xsd:string" use="optional" />
  </xsd:complexType>
</xsd:element>

<xsd:element name="job-step">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="step-scheduling" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="jndi-name" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

```

        <xsd:element ref="checkpoint-algorithm-ref" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="results-ref" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="batch-data-streams" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="props" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="optional" />
    <xsd:attribute name="application-name" type="xsd:string" use="optional" />
</xsd:complexType>
</xsd:element>

<xsd:element name="prop">
    <xsd:complexType>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="value" type="xsd:string" use="required" />
    </xsd:complexType>
</xsd:element>

<xsd:element name="props">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="prop" maxOccurs="unbounded" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="returncode-expression">
    <xsd:complexType>
        <xsd:attribute name="step" type="xsd:string" use="required" />
        <xsd:attribute name="operator" type="xsd:string" use="required" />
        <xsd:attribute name="value" type="xsd:string" use="required" />
    </xsd:complexType>
</xsd:element>

<xsd:element name="step-scheduling">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="returncode-expression" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="condition" type="xsd:string" use="optional" />
    </xsd:complexType>
</xsd:element>

<xsd:element name="step-scheduling-criteria">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="scheduling-mode" minOccurs="1" maxOccurs="1" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>

```

xJCL sample for a compute intensive job

The xJCL sample is a generic compute intensive sample.

```

<?xml version="1.0" encoding="UTF-8" ?>
<job name="OpenGrid" class="xyz" accounting="accounting info" default-application-name="tryit"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <job-scheduling-criteria>
        <required-capability expression="someExpression" />
        <required-capability expression="anotherExpression" />
    </job-scheduling-criteria>

    <substitution-props>
        <prop name="PATH" value="C:\\windows;C:\\java\\jre\\bin" />
    </substitution-props>

    <job-step name="Step1" application-name="tryit">

```

```

<env-entries>
<env-var name="PATH" value="{PATH}" />
<env-var name="CLASSPATH" value="C:\\windows" />
</env-entries>

<exec executable="java">
<arg line="command line args here" />
<arg line=" and more command line args here" />
</exec>

</job-step>

</job>

```

XML schema for a compute intensive job

The following example shows the XML schema for a compute-intensive job:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="classname" type="xsd:string" />
  <xsd:element name="jndi-name" type="xsd:string" />

  <xsd:element name="required-capability">
    <xsd:complexType>
      <xsd:attribute name="expression" type="xsd:string" use="required" />
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="substitution-props">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="prop" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="job-scheduling-criteria">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="1" ref="required-capability" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="job">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="jndi-name" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="job-scheduling-criteria" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="substitution-props" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="job-step" maxOccurs="unbounded" minOccurs="1" />
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required" />
      <xsd:attribute name="class" type="xsd:string" use="optional" />
      <xsd:attribute name="accounting" type="xsd:string" use="optional" />
      <xsd:attribute name="default-application-name" type="xsd:string" use="optional" />
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="job-step">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="classname" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="props" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="optional" />
      <xsd:attribute name="application-name" type="xsd:string" use="optional" />
    </xsd:complexType>
  </xsd:element>

```

```

<xsd:element name="prop">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="value" type="xsd:string" use="required" />
  </xsd:complexType>
</xsd:element>

<xsd:element name="props">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="prop" maxOccurs="unbounded" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

</xsd:schema>

```

xJCL sample for a native execution job

The native execution sample is for a single-step job, which are the only type of jobs that are supported.

```

<?xml version="1.0"?>
<job name="GridUtilitySample" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <job-step name="Step1" application-name="tryit">
    <env-entries>
      <env-var name="PATH" value="/opt/IBM/WebSphere/AppServer/java/jre/bin"/>
      <env-var name="CLASSPATH" value="/user/classes"/>
    </env-entries>
    <exec executable="java">
      <arg line="GridUtilitySample"/>
    </exec>
  </job-step>
</job>

```

XML schema for a native execution job

This XML schema example can be used with xJCL to implement a native execution job.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="arg">
    <xsd:complexType>
      <xsd:attribute name="line" type="xsd:string" use="required" />
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="required-capability">
    <xsd:complexType>
      <xsd:attribute name="expression" type="xsd:string" use="required" />
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="substitution-props">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="prop" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="env-entries">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="env-var" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="env-var">
    <xsd:complexType>

```

```

        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="value" type="xsd:string" use="required" />
    </xsd:complexType>
</xsd:element>

<xsd:element name="exec">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="arg" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="executable" type="xsd:string" use="required" />
    </xsd:complexType>
</xsd:element>

<xsd:element name="job-scheduling-criteria">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" ref="required-capability" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="job">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="job-scheduling-criteria" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="substitution-props" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="job-step" maxOccurs="unbounded" minOccurs="1" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="class" type="xsd:string" use="optional" />
        <xsd:attribute name="accounting" type="xsd:string" use="optional" />
        <xsd:attribute name="default-application-name" type="xsd:string" use="optional" />
    </xsd:complexType>
</xsd:element>

<xsd:element name="job-step">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="env-entries" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="exec" minOccurs="1" maxOccurs="1" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="optional" />
        <xsd:attribute name="application-name" type="xsd:string" use="optional" />
    </xsd:complexType>
</xsd:element>

<xsd:element name="prop">
    <xsd:complexType>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="value" type="xsd:string" use="required" />
    </xsd:complexType>
</xsd:element>

</xsd:schema>

```

CommandRunner utility job step

Use the CommandRunner utility job step to run shell command lines as job steps. The shell command lines can include shell commands, shell scripts, and compiled programs.

The CommandRunner utility runs the specified shell command line in an operating system process. Standard output and standard error streams are captured and written to the job log. The command-line return code is captured and set as the step return code. If the job step is canceled, the return code is -8.

Command syntax

Use the following syntax for the CommandRunner utility:

```

<job-step name={step_name}>
<classname>com.ibm.websphere.batch.utility.CommandRunner</classname>
{job_step_properties}
</job-step>

```

For example, the following job step code runs a command-line Java program:

```

<job-step name="RunJava">
<classname>com.ibm.websphere.batch.utility.CommandRunner</classname>
<props>
<prop name="com.ibm.websphere.batch.cmdLine"
value="java.exe com.ibm.websphere.batch.samples.TestCase" />
<prop name="CLASSPATH" value="{user.dir}\testcases;{user.dir}\bin" />
<prop name="Path" value="{java.home}\bin;{env:Path}" />
</props>
</job-step>

```

Required job step property

The following property is required for the CommandRunner utility job step.

com.ibm.websphere.batch.cmdLine

Specifies the command-line invocation, including arguments.

For example, run the java.exe file as the command line:

```

<prop name="com.ibm.websphere.batch.cmdLine"
value="java.exe com.ibm.websphere.batch.samples.TestCase" />

```

Optional job step properties

The following properties are optional for the CommandRunner utility job step.

com.ibm.websphere.batch.workingDir

Specifies working directory in which the specified command-line runs.

For example, enable /tmp as the working directory on Linux:

```

<prop name="com.ibm.websphere.batch.workingDir" value="/tmp" />

```

The default is undefined.

com.ibm.websphere.batch.debug

Specifies true or false to indicate whether the command line runs in debug mode or not. Debug mode prints debug messages to assist you in resolving problems with variable substitution and other issues that prevent command lines from running correctly.

For example, enable the debug mode:

```

<prop name="com.ibm.websphere.batch.debug" value="true" />

```

The default is false.

com.ibm.websphere.batch.expansion

Specifies true or false to enable or disable job step property expansion, also known as property substitution. Disable this feature to improve step performance for steps with large property maps that do not depend on property expansion.

For example, disable property expansion:

```

<prop name="com.ibm.websphere.batch.expansion" value="false" />

```

The default is true.

com.ibm.websphere.batch.shell.executor

Specifies the shell executor command. Use this property to specify a custom shell as your command-line executor.

For example, specify a custom shell as fastshell:

```

<prop name="com.ibm.websphere.batch.shell.executor" value="fastshell" />

```

The default is the value of the `com.ibm.websphere.batch.command.runner.shell.executor` system property.

com.ibm.websphere.batch.shell.failure.rc

Specifies the shell execution failure return code. This return code is returned by the shell executor to indicate the specified command line can not be run. When the shell executor return code matches the shell failure return code value, the CommandRunner utility job sets the job step return code to the value of `com.ibm.websphere.batch.step.failure.rc` property.

For example, the shell returns -1 when the specified command line fails to run:

```
<prop name="com.ibm.websphere.batch.shell.failure.rc" value="-1" />
```

The default is the value of the `com.ibm.websphere.batch.command.runner.shell.failure_rc` system property.

com.ibm.websphere.batch.step.failure.rc

Specifies the job step return code when the command line fails to execute. This job step return code is returned when the shell executor return code matches the value of `com.ibm.websphere.batch.shell.failure.rc`.

For example, the job step returns -1 when the shell fails to run:

```
<prop name="com.ibm.websphere.batch.step.failure.rc" value="-1" />
```

System properties

All the CommandRunner utility system properties are optional.

com.ibm.websphere.batch.command.runner.shell.executor

Specifies the default value for the `com.ibm.websphere.batch.shell.executor` job step property.

Windows The default is `cmd.exe /c`.

AIX **HP-UX** **Solaris** **Linux** **z/OS** The default is `sh -c`.

com.ibm.websphere.batch.command.runner.shell.failure_rc

Specifies the default value for the `com.ibm.websphere.batch.shell.failure.rc` job step property.

Windows The default is 1.

AIX **HP-UX** **Solaris** **Linux** **z/OS** The default is 127.

Property substitution

The CommandRunner utility job step supports property substitution for expansion. You can do expansion using xJCL substitution properties, Java system properties, and process variables as demonstrated in the following examples.

xJCL substitution property

The following example substitutes the `testcase.dir` xJCL substitution property as a working directory.

```
<prop name="com.ibm.websphere.batch.workingDir" value="${testcase.dir}" />
```

Java system property

The following example substitutes the `user.dir` Java system property in CLASSPATH process variable:

```
<prop name="CLASSPATH" value="${user.dir}/classes" />
```

Process variable

The following example substitutes the `PATH` process variable in the new value of the `PATH` process variable:

```
<prop name="PATH" value="/tmp:${env:PATH}" />
```

Important: Substitution properties are treated as Java system properties, if they are displayed in the Java system properties list. Otherwise, they are treated as xJCL substitution properties. Process variable substitution is denoted by the special `${env:<variable name>}` syntax.

Process variables

The CommandRunner utility adds all job step properties, after substitution, to the process variable pool for the process in which the specified command-line runs.

WSGrid properties file examples

Use the WSGrid properties file examples in your job submissions.

Example: Jobs from repository properties file

WSGrid can submit jobs stored in the job repository. You can use a repository properties file to specify the saved job definition and its substitution properties.

If the PostingsSample job is stored in the repository under the name PostingsSample, you can use the following properties file, named WSGrid.repo-postings.job, to specify the saved job definition and its substitution properties:

```
# This specifies which job from the repository to submit.
repository-job=PostingsSample

# The following substitution property values will be passed to the job.
substitution-prop.wsbatch.count=5
substitution-prop.checkpoint=timebased
substitution-prop.checkpointInterval=15
substitution-prop.postingsDataStream=${was.install.root}/temp/postings"
```

The WSGrid invocation to submit the job described by the preceding properties file is:

```
WSGrid WGrid.cntl WSGrid.repo-postings.job
```

Example: Compute-intensive properties file

The example file WSGrid.ci.job illustrates the properties necessary to define the single step of the SimpleCI compute intensive job.

```
# Specify name by which this job is known job-name=SimpleCIEar

# Note, if not specified application-name defaults to job-name.

# This is the jndi name of the Compute Intensive Controller system SLSB.
controller-jndi-name=ejb/com/ibm/ws/ci/SimpleCIEJB

# This is the name of the compute intensive POJO class that implements the
# business logic
ci-class-name=com.ibm.websphere.ci.samples.SimpleCIWork
# The following properties are passed to SimpleCIWork
prop.calculationTimeInSecs=30
prop.outputFileName=/temp/ci.out
```

The WSGrid invocation to submit the job described by the preceding properties file is:

```
WSGrid WGrid.cntl WSGrid.ci.job
```

Example: Transactional batch properties file

The example file WSGrid.batch.job illustrates the properties necessary to define the first step of the PostingsSample batch job.


```

# Specify name by which this job is known job-name=PostingsSample

# Use default job class - uncomment to specify custom class
# job-class=<enter class name here>

# This is the JNDI name of the BatchController system SLSB.
controller-jndi-name=ejb/com/ibm/websphere/samples/PostingsJob

# Specify name of target Java(TM) Platform, Enterprise Edition (Java EE)application.
application-name=PostingsSampleEar

# A time-based checkpoint algorithm is provided.

checkpoint-algorithm=com.ibm.wsspi.batch.checkpointalgorithms.timebased

# Use the interval property of the algorithm, specified in seconds, to decide
# how often to commit the global transaction when invoking a batch job step.
checkpoint-algorithm.interval=10

# This is a logical JNDI name for the batch step; it has to match the ejb-reference declared
# in the system SLSB for this batch step entity bean.

batch-bean-jndi-name=ejb/DataCreationBean

# This is the output bds used by the DataCreationBean.
# The logical name expected by the DataCreationBean is 'myoutput'

bds.myoutput=com.ibm.websphere.samples.PostingOutputStream

# Specify the implementation class and bds input property named 'FILENAME', expected
# by this bds class. Change the value of the 'FILENAME' property to a path
# in the filesystem to write the postings output file.

bds.myoutput.FILENAME=/root/bds/sample/myostingsfile
# Generic properties can be passed to the Batch Step. The DataCreationBean step uses
# this property to control how many postings to create in the file associated with
# PostingsOutputStream.
prop.name.wsbatch.count=5

```

Example: Restart job properties file

Use the WSGrid utility to restart jobs that are in the restartable state by using the job ID of the restartable job of the properties file.

For example, if PostingSample:0001 ended in a restartable state, you can restart it by using the restart.props file, which contains restart-job=postingSample:0001.

From the command prompt, issue this command:

```
>wsgrid.sh cntl.props restart.prop
```

The job specified in restart-prop is restarted.

Using the WSGrid utility to create restart.prop

Specify a file name using the third argument, as shown in the following:

```
>wsgrid.sh cntl.props job.props restart.prop
```

If a job ends in a restartable state, a file named restart.prop is created and the job ID is written as shown in the example.

Example: xJCL file

Use the WSGrid invocation to submit a job for a pre-existing xJCL file.

The following invocation includes a pre-existing xJCL file `job.xml` and a sample control file `WSGrid.cntl`:

```
WSGrid WSGrid.cntl job.xml
```

Example: Control file

A control file can contain the following properties.

```
# host of my job scheduler
scheduler-host=zwasc013.lab.ibm.com
# http port of my job scheduler server
scheduler-port=9182
# user id of job submitter
submitter-userid=jobman
# job submitter password
submitter-password=pass2wrd

# enable debug
debug=true
# increase timeout to 8 seconds per message
timeout=8000
```

Example control file with encoded password

The password in the control file can be encoded for security purposes by using the `PropFilePasswordEncoder` utility. For file `WSGrid.cntl`, which contains the control properties from the previous example, the encoding utility is invoked as:

```
PropFilePasswordEncoder WSGrid.cntl submitter-password
```

After the `PropFilePasswordEncoder` is run against `WSGrid.cntl` file, the value of the `submitter-password` is encoded in the `WSGrid.cntl` file. For example:

```
submitter-password={xor}0TAWPT4tbj4=
```

Chapter 3. Scripting batch applications

After you install the product, you can use scripts to complete various tasks.

jobrecovery.batl.sh batch script

You can use the jobrecovery script to enable a secondary site to take over when the primary site fails.

Purpose

In this scenario, each site has a separate WebSphere cell where an active job is running on the primary site, and a stopped or job scheduler is on the secondary site. The batch cells on the primary and secondary sites are expected to be clones of one another with the same topography and the same names. They share the same database, not replicas of a database.

Location

The jobrecovery script is located either in the *app_server_root/bin* directory or in the *app_server_root/profiles/{LRS_profiles}/bin* directory on the designated LRS backup server. If the script is run from the *app_server_root/bin* directory, the *profileName* option must be specified. Otherwise, do not specify the *profileName* option. The user name and password options are required if security is enabled on the server which is running the LRS. Run on the backup cell after the primary cell is shut down. After it is run, the backup cell can be started.

Usage

When the primary site fails, you are instructed to follow a takeover procedure that includes the following steps:

1. Ensure that all batch servers on the primary site are stopped.
2. Run the jobrecovery script on the secondary site.
3. Start the job scheduler on the secondary site.

```
./jobRecovery.sh server [options]
```

Options

-profileName *profile*

Use this optional parameter to specify the stand-alone application server profile name when the script is run from the *app_server_root/bin* directory.

-username *authentication username*

Use this optional parameter to specify a user name.

-password *authentication password*

Use this optional parameter to specify an authentication password for the user name.

Example

```
./jobRecovery.sh server1 -profileName AppSrv01 -username wsadmin -password wspassword
```

uteconfig.batl.sh batch script

You can use the uteconfig script to configure the unit test environment (UTE). Run the script to create a job scheduler configuration on the WebSphere Application Server profile that is used in Rational Application Developer.

Purpose

Use the `uteconfig` script to create a job scheduler configuration on the application server profile. The script creates Derby resources, deploys the job scheduler application (`LongRunningScheduler.ear`), creates a work manager, and otherwise configures a UTE on the profile. The UTE provides an environment for testing applications developed with Rational Application Developer.

Location

The `uteconfig.bat|.sh` script is located in the `bin` directory of an application server profile; for example, `app_server_root/profiles/AppSrv01/bin`. You must have a stand-alone application server profile to run the script.

The `uteconfig.bat|.sh` script also is located in the main `bin` directory of the product; for example, `app_server_root/bin`.

Usage

1. Open a command prompt on a `bin` directory that has the `uteconfig.bat|.sh` script.
It is recommended to run the `uteconfig` script from the `bin` directory of the application server profile. To run the `uteconfig` script successfully from the main `bin` directory, `app_server_root/bin`, your product installation must have only one application server profile.
2. Run the `uteconfig.bat|.sh` script.

The script takes about one minute to run and provides progress messages.

Options

None

Example

```
./uteconfig.sh
```

configCGSharedLib.py batch script

You can use the `configCGSharedLib.py` Jython script to assign the shared library to the job scheduler.

Purpose

The `configCGSharedLib.py` script is provided with the product. The `configCGSharedLib.py` script assigns the shared library to the job scheduler.

Location

At installation, the `configCGSharedLib.py` script is copied onto the installation target in the `app_server_root/bin` directory.

Usage

To run the `configCGSharedLib.py` script with the `wsadmin` utility, use this command:

```
wsadmin -lang jython -f configCGSharedLib.py <option>
```

You might have to modify the `wsadmin` command to `wsadmin.sh` or `wsadmin.bat`, depending on your operating system environment.

To see a list of all available operations, use the following command:

```
wsadmin -lang jython -f configCGSharedLib.py --help
```

removePGC.py batch script

You can use the `removePGC.py` Jython script to remove the common batch container from your deployment target.

Purpose

The `removePGC.py` script is provided with the product. The `removePGC.py` script removes the common batch container from your deployment target or removes it when your deployment target is the only target.

Location

At installation, `removePGC.py` is copied onto the installation target machines in the `app_server_root/bin` directory.

Usage

To run `removePGC.py` script with the `wsadmin` utility, use this command:

```
wsadmin -lang jython -f removePGC.py <option>
```

You might have to modify the `wsadmin` command to `wsadmin.sh` or `wsadmin.bat`, depending on your operating system environment.

To see a list of all available operations, use the following command:

```
wsadmin -lang jython -f removePGC.py --help
```

Operations

Use the following option with this command:

--list

Lists the targets that have the common batch container.

Example

Use following command to list the targets which have the common batch container:

```
wsadmin -lang jython -f removePGC.py --list
```

For example:

```
>> wsadmin -lang jython -f removePGC.py --list
INFO: Grid Execution Environment was found on following targets:
      cell=myCell,cluster=Endpoint1
      cell=myCell,cluster=Endpoint2
      cell=myCell,node=myNode01,server=server1
```

redeployLRS.py batch script

You can use the `redeployLRS.py` Jython script to redeploy the job scheduler on your deployment target.

Purpose

The `redeployLRS.py` script is provided with the product. The `redeployLRS.py` script redeploys the job scheduler on your deployment target.

Location

At installation, `redeployLRS.py` is copied onto the installation target in the `app_server_root/bin` directory.

Usage

To run `redeployLRS.py` script with the `wsadmin` utility, use this command:

```
wsadmin -lang jython -f redeployLRS.py <option>
```

Modify the `wsadmin` command to `wsadmin.sh` or `wsadmin.bat`, depending on your operating system environment.

To see a list of all available operations, use the following command:

```
wsadmin -lang jython -f redeployLRS.py --help
```

wsgriidConfig.py batch script

Use the `wsgriidConfig.py` script to configure the three steps that are required for configuring the external scheduler interface.

Purpose

The `wsgriidConfig.py` script performs the following three steps required for configuring the external scheduler interface:

1. Installs `JobSchedulerMDI.ear`.
2. Configures the service integration bus.
3. Configures JMS artifacts.

Location

At installation, `wsgriidConfig.py` is copied onto the installation target machines in the `app_server_root/bin` directory.

Example

To configure WSGrid on a scheduler cluster, use the following command:

```
wsadmin.sh -user username -password userpassword -f ../bin/wsgriidConfig.py  
-install -cluster clusterName -providers providerList
```

where *clusterName* is the scheduler cluster name.

To configure WSGrid on a single scheduler server, use the following command:

```
wsadmin.sh -user username -password userpassword -f ../bin/wsgriidConfig.py  
-install -node nodeName -server serverName -providers providerList
```

where *nodeName* is the node name of the scheduler server.

providerList identifies a list of provider endpoints in the format `hostname1,portnumber1[;hostname2,portnumber2...]`, where `portnumber` identifies the `SIB_ENDPOINT_ADDRESS` or `SIB_ENDPOINT_SECURE_ADDRESS` port of the scheduler server, and `hostname1` and `hostname2` identifies the host name of the servers in the scheduler cluster.

To remove WSGrid configurations, use the following commands:

```
wsadmin.sh -user username -password userpassword -f ../bin/wsgriidConfig.py  
-remove -cluster clusterName
```

```
wsadmin.sh -user username -password userpassword -f ../bin/wsgridConfig.py
-remove -node nodeName -server serverName
```

JobSchedulerCommands command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure the job scheduler with the wsadmin tool. The commands and parameters in the JobSchedulerCommands command group can be used to manage configuration attributes and custom properties.

Use the following commands to manage the job scheduler:

- “showJobSchedulerAttributes”
- “modifyJobSchedulerAttribute” on page 86
- “createJobSchedulerProperty” on page 86
- “modifyJobSchedulerProperty” on page 87
- “removeJobSchedulerProperty” on page 88
- “listJobSchedulerProperties” on page 88

showJobSchedulerAttributes

The showJobSchedulerAttributes command shows all configuration attributes of the job scheduler.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns a list of all attributes of the job scheduler.

Batch mode example usage

- Using Jacl
`$AdminTask showJobSchedulerAttributes`
- Using Jython
`AdminTask.showJobSchedulerAttributes()`

Interactive mode example usage

- Using Jacl:
`$AdminTask showJobSchedulerAttributes`
- Using Jython:
`AdminTask.showJobSchedulerAttributes()`

modifyJobSchedulerAttribute

The modifyJobSchedulerAttribute command modifies a configuration attribute of the job scheduler.

Target object

None

Required parameters

-name

Specifies the name of the attribute to modify. (String)

The following attributes are supported.

1. datasourceJNDIName (default value is jdbc/lrsched)
2. databaseSchemaName (default value is LRSSHEMA)
3. deploymentTarget (default value is none)
4. endpointJobLogLocation (default value is \${GRID_JOBLOG_ROOT})
5. enableUsageRecording (default value is false)
6. enableUsageRecordingZOS (default value is false)

Optional parameters

-value

Specifies the value of the attribute. (String) If not specified, the default value for the respective attributes is assigned.

Return value

The command returns the job scheduler object ID.

Batch mode example usage

- Using Jacl:

```
$AdminTask modifyJobSchedulerAttribute {-name datasourceJNDIName -value "jdbc/ds"}
```
- Using Jython:

```
AdminTask.modifyJobSchedulerAttribute(['-name datasourceJNDIName -value jdbc/ds'])
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask modifyJobSchedulerAttribute {-interactive}
```
- Using Jython:

```
AdminTask.modifyJobSchedulerAttribute(['-interactive'])
```

createJobSchedulerProperty

The createJobSchedulerProperty command creates custom properties for the job scheduler.

Target object

None

Required parameters

-name

Specifies the name of the custom property to create. (String)

-value

Specifies the value of the custom property. (String)

Optional parameters

-description

Specifies the description of the custom property. (String)

Return value

The command returns the properties object ID.

Batch mode example usage

- Using Jacl:
`$AdminTask createJobSchedulerProperty {-name bjsProp1 -value "bjsprop1"}`
- Using Jython:
`AdminTask.createJobSchedulerProperty('[-name bjsProp1 -value bjsprop1]')`

Interactive mode example usage

- Using Jacl:
`$AdminTask createJobSchedulerProperty {-interactive}`
- Using Jython:
`AdminTask.createJobSchedulerProperty('[-interactive]')`

modifyJobSchedulerProperty

The `modifyJobSchedulerProperty` command modifies custom properties for the job scheduler.

Target object

None

Required parameters

-name

Specifies the name of the custom property to modify. (String)

-value

Specifies the value of the custom property. (String)

Optional parameters

-description

Specifies the description of the custom property. (String)

Return value

The command returns the properties object ID.

Batch mode example usage

- Using Jacl:
`$AdminTask modifyJobSchedulerProperty {-name bjsProp1 -value "bjsprop1"}`
- Using Jython:
`AdminTask.modifyJobSchedulerProperty('[-name bjsProp1 -value bjsprop1]')`

Interactive mode example usage

- Using Jacl:
\$AdminTask modifyJobSchedulerProperty {-interactive}
- Using Jython:
AdminTask.modifyJobSchedulerProperty('[-interactive]')

removeJobSchedulerProperty

The removeJobSchedulerProperty command removes custom properties of the job scheduler.

Target object

None

Required parameters

- name**
Specifies the name of the custom property to remove. (String)

Optional parameters

None

Return value

The command returns the properties object ID.

Batch mode example usage

- Using Jacl:
\$AdminTask removeJobSchedulerProperty {-name bjsProp1}
- Using Jython:
AdminTask.removeJobSchedulerProperty('[-name bjsProp1]')

Interactive mode example usage

- Using Jacl:
\$AdminTask removeJobSchedulerProperty {-interactive}
- Using Jython:
AdminTask.removeJobSchedulerProperty('[-interactive]')

listJobSchedulerProperties

The listJobSchedulerProperties command lists all of the custom properties the job scheduler.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns a list of all of the custom properties of the job scheduler.

Batch mode example usage

- Using Jacl:
`$AdminTask listJobSchedulerProperties`
- Using Jython:
`AdminTask.listJobSchedulerProperties()`

Interactive mode example usage

- Using Jacl:
`$AdminTask listJobSchedulerProperties`
- Using Jython:
`AdminTask.listJobSchedulerProperties()`

Chapter 4. Developing batch applications

This section covers such areas as a procedure for developing batch applications, xJCL elements, and sample batch applications.

Transactional batch and compute-intensive batch programming models

The product provides a transactional batch programming model and a compute-intensive programming model.

Both the transactional batch and compute-intensive programming models are implemented as Java objects. They are packaged into an enterprise archive (EAR) file for deployment into the application server environment. The individual programming models provide details on how the life cycle of the application and jobs submitted to it are managed by the grid endpoints. Central to all batch applications is the concept of a job to represent an individual unit of work to be run.

You can mix transactional batch, compute intensive, and native execution job steps. The run time uses a controller that is the same for every job, regardless of the type of steps that the job contains. The controller runs appropriate logic for the step, whether the step is a batch, compute-intensive, or native execution step. These different job step types can also be run in parallel.

The Java Platform, Enterprise Edition (Java EE) applications that the application server hosts typically perform short, lightweight, transactional units of work. In most cases, an individual request can be completed with seconds of processor time and relatively little memory. Many applications, however, must complete batch work that is computational and resource intensive. The batch function extends the application server to accommodate applications that must perform batch work alongside transactional applications. Batch work might take hours or even days to finish and uses large amounts of memory or processing power while it runs.

COBOL container overview

The COBOL container enables COBOL modules to be loaded into the batch address space, and they are invoked directly.

The container can be created and destroyed multiple times within the lifecycle of an application server. Each container is created with a Language Environment[®] (LE) enclave separate from the server. The container is assured of a clean LE each time it is created.

Java programs can pass parameters into COBOL and retrieve the results. The COBOL call stub generator tool. is provided to create the Java call stubs and data bindings based on the data and linkage definitions in the COBOL source. Additionally, JDBC Type 2 connections created by the Java program can be shared with the COBOL program under the same transactional context.

Value of the COBOL container

The product provides a comprehensive execution environment for Java batch processing. Part of the design of batch support is the integration with other information processing. COBOL has been a part of batch processing since the very early days of computers, and there is significant investment in COBOL assets. The COBOL container provides a means of direct integration of COBOL into Java processing.

Programming restrictions for COBOL programs

COBOL programs that run in the product environment have the following programming restrictions:

- User-defined error handlers are not allowed.
- Explicit transaction control is not allowed: for example, no COMMIT or ROLLBACK.
- DB2 special registers, such as SOLID or SCHEMA, cannot be set.
- DD access requires user code to dynamically allocate or deallocate the DD.
- Java invocation of ENTRY labels is not supported.
- COBOL invocation of Java code is not supported

COBOL compilation requirements

You must compile all COBOL modules that you use in the environment with the following options:

dll COBOL modules must be in DLL format.

rent COBOL modules must contain reentrant code.

thread
COBOL modules must be thread safe.

outdd(WCGILOUT)
Required for COBOL DISPLAY output to appear in product job logs.

SQL('ATTACH(RRSAP)')
Required for DB2 access.

You can specify these options in the COBOL source:

```
cb1 dll,lib,rent,thread,outdd(wcgilout)
```

You can also specify these options as inputs to the COBOL compiler. For example, enter the following on the cob2 command line:

```
cob2 -c -bdll,rent,thread,lib,list '-qOUTDD(WCGILOUT)' sample.cb1
```

Restrictions for the COBOL container

The product hosting the COBOL container must be configured with a workload profile of ISOLATE.

For information about how to configure the workload profile, read about ORB services advanced settings on the z/OS platform.

JDBC data source restrictions for the COBOL container

The restrictions listed in this section apply to any JDBC data source that contains DB2 type-2 connections that are shared between the Java and COBOL code through `ILContainer.setDB2Connection`. [Also, specify the object type for `ILContainer.setDB2Connection`.]

These restrictions ensure that any DB2 data constructs that are opened by the COBOL code and that persist across COMMIT boundaries, such as WITH HOLD CURSORS, are properly cleaned up when the JDBC connection is closed.

Connection pooling must be DISABLED. You can disable connection pooling by using the `disableWASConnectionPooling` property. To set this property using the administrative console, click **Resources > JDBC > Data sources > data_source_name > Custom properties > New**. Add a custom property named `disableWASConnectionPooling` and set the value to true.

Connection sharing must be set to UNSHARED. Connection sharing for the data source can be set to UNSHARED by using the `globalConnectionTypeOverride` property. To set this property using the administrative console, click **Resources > JDBC > Data sources > *data_source_name* > Connection pool properties > Connection pool custom properties > New**. Add a custom property named `globalConnectionTypeOverride` and set the value to `unshared`.

For more information about these properties, read about [Tuning connection pools](#).

Developing COBOL container batch applications

You can use the COBOL container within the product to invoke COBOL modules from a Java-based batch application.

Before you begin

Read the [COBOL container overview](#) topic for more detailed information.

About this task

The COBOL container provides a means of direct integration of COBOL into Java batch processing.

Procedure

1. Create a COBOL call stub Java class
If you want to call a COBOL module from a Java batch plain old Java object (POJO) in the batch environment, you must first create a COBOL call stub Java class.
2. Compile a COBOL call stub Java class
You can compile COBOL call stubs with the `marshal.jar` and `ilcontainer.jar` files on the classpath. The `marshal.jar` and `ilcontainer.jar` files are included with the COBOL call stub generator tool.
3. Invoke a COBOL call stub Java class.
Invoke the Java call stub from your batch application. Read the [COBOL call stub Java class usage example](#) topic for more information.
4. Optional: Dynamically update a COBOL module
You can dynamically update a COBOL module without having to restart the application server.

Creating a COBOL call stub Java class

If you want to call a COBOL module from a Java batch plain old Java object (POJO) in the batch environment, you must first create a COBOL call stub Java class.

Procedure

1. Install the COBOL call stub generator.
The COBOL call stub generator tool compressed file can be found in the `<was-home>/lib` directory of your batch installation. Extract the file anywhere on your workstation.

Note: The COBOL call stub generator tool requires Java 1.6.

2. Create a COBOL call stub generator properties file for your system. Read the [Creating a call stub generator configuration file](#) topic for more information.
3. Use the COBOL call stub generator tool to create a COBOL call stub Java class; for example:

```
$ cd COBOLCallStubGenerator
$ java -jar lib/COBOLCallStubGenerator.jar testcases/Sample01.cb1 \
    -configFile csg.properties \
    -callStubPackage com.ibm.ws.batch.ilc.sample \
    -callStubClass Sample
```

The COBOL call stub generator tool can also be invoked from within Rational Application Developer. Read the Generating COBOL call stubs topic for more information about the tool.

The generated code is written to the `src` directory of the Rational Application Developer workspace and project directory you specify on the `WorkSpace` and `EclipseProjectName` properties in the `csg.properties` file, which is found in the COBOL call stub generator install directory (`/COBOLCallStubGenerator`).

The COBOL call stub generator tool then writes `Sample.java` to the following directory:

```
${WorkSpace}/${EclipseProjectName}/src/com/ibm/ws/batch/ilc/sample
```

and any data binding classes for the linkage section variable used on the `PROCEDURE` statement to the following directory:

```
${WorkSpace}/${EclipseProjectName}/src/com/ibm/ws/batch/ilc/sample/parameters
```

Compiling COBOL call stub Java classes

COBOL call stub Java classes must be compiled with the `marshal.jar` and `ilcontainer.jar` files on the class path. The `marshal.jar` and `ilcontainer.jar` files are included with the COBOL call stub generator tool.

Procedure

1. To compile the data bindings classes, use the following example:

```
$ CSG_LIB=/COBOLCallStubGenerator/lib
$ javac -cp ./;$CSG_LIB/marshal.jar;$CSG_LIB/ilcontainer.jar \
com/ibm/ws/batch/ilc/sample/parameters/*.java
```

2. To compile the call stub, use the following example:

```
$ javac -cp ./;$CSG_LIB/marshal.jar;$CSG_LIB/ilcontainer.jar \
com/ibm/ws/batch/ilc/sample/*.java
```

Note: Add the `marshal.jar` and `ilcontainer.jar` files to your IBM Rational Application Developer for WebSphere Software product build path to compile the stub and data bindings classes.

Dynamically updating a COBOL module

You can dynamically update a COBOL module without having to restart the batch application server.

About this task

The COBOL module DLL is loaded by the `ILContainer` upon first invocation of the COBOL procedure, and is released by the `ILContainer` when the `ILContainer` is destroyed, which is at the end of the batch step.

This function enables you to dynamically update the COBOL module between batch jobs or steps without having to restart the batch application server. The updated COBOL module is dynamically loaded by the `ILContainer` that is created during the next batch step.

Procedure

1. Update the COBOL module.
2. Run the batch application.

For more information about running batch applications, read about the batch programming model.

COBOL call stub Java class usage example

The COBOL call stub Java class usage example shows how a batch application can invoke a COBOL procedure using the COBOL container.

In the example, the events take place in the following order:

1. The COBOL container is created.

2. The COBOL procedure call stub is created.
3. The parameter data is set into the COBOL procedure call stub.
4. A shared DB2 connection is set on the container.
5. The COBOL procedure is invoked using the container.

Steps 1, 2 and 5 are the minimum steps required to invoke a COBOL procedure using the container. Steps 3 and 4 are optional. Step 3 is necessary only if the COBOL procedure receives parameters, and step 4 is necessary only if the COBOL procedure accesses DB2.

```
import com.ibm.websphere.batch.ilc.ILContainerFactory;
import com.ibm.websphere.batch.ilc.ILContainer;
import com.ibm.websphere.batch.ilc.ILContainerException;
import com.ibm.websphere.batch.ilc.ILProcedureException;

import com.ibm.ws.batch.ilc.sample.Sample;
import com.ibm.ws.batch.ilc.sample.parameters.SampleDataBinding;

import javax.naming.InitialContext;
import javax.sql.DataSource;
import java.sql.Connection;

try {
    // Create the container.
    ILContainer container = ILContainerFactory.getFactory().create();

    // Create target procedure using call stub.
    Sample proc = new Sample();

    // Set parameters.
    SampleDataBinding binding = proc.getSampleDataBinding();
    binding.setCDummy("foo");
    binding.setIlen((short)employeeNumber);

    // Set db2 connection for use by COBOL (if necessary).
    // InitialContext ic = new InitialContext();
    // DataSource datasourceType2 = (DataSource) ic.lookup(jdbcJndi);
    // Connection connectionType2 = datasourceType2.getConnection();
    // container.setDB2Connection(connectionType2);

    // Invoke the COBOL procedure.
    int rc = container.invokeProcedure(proc);
}
catch (ILProcedureException) {
    ...
}
catch (ILContainerException) {
    ...
}
```

COBOL RETURNING, RETURN-CODE, getReturnValue, and getReturnCode parameters

If the COBOL module specifies a RETURNING parameter, it can be retrieved from the stub using the `stub.getReturnValue()` method. The method returns the Java object representation of the RETURNING parameter, which is the data bindings class generated by the COBOL call stub generator tool.

See the following information about the COBOL module and its parameters:

- If the COBOL module does not specify a RETURNING parameter, then the `getReturnValue()` method is not generated for the stub.
- If the COBOL module sets the RETURN-CODE special register, its value can be retrieved from the stub using the `stub.getReturnCode()` method. The method returns an `int`.
- If the COBOL module does not set the RETURN-CODE special register, then the `getReturnCode()` method always returns 0.

- If the COBOL module specifies a RETURNING parameter, and sets the RETURN-CODE special register, then COBOL ignores the RETURN-CODE register. In this case, the `stub.getReturnValue()` method returns the RETURNING parameter, and the `stub.getReturnCode()` method always returns 0.

Table 34. `stub.getReturnValue()` and `stub.getReturnCode()` methods return summary. This table summarizes what is returned by the `stub.getReturnValue()` and `stub.getReturnCode()` methods based on what is specified in the COBOL code.

COBOL specifies:	<code>stub.getReturnValue()</code>	<code>stub.getReturnCode()</code>
RETURNING	The RETURNING parm	0
RETURN-CODE	(not generated)	RETURN-CODE
Both RETURNING and RETURN-CODE	The RETURNING parm	0
Neither	(not generated)	0

COBOL container for batch troubleshooting

If you encounter problems when using COBOL container for batch troubleshooting, there are a number of options that are available to you.

Debug trace

To enable tracing of the COBOL container code, add the following to the Java trace specification:

```
com.ibm.ws.batch.ilc.*=all
com.ibm.websphere.batch.ilc.*=all
```

Performance trace

To monitor performance of COBOL container invocations, activate the following Java trace specification:

```
com.ibm.ws.batch.ilc.impl.LEChildEnvironment.invokeProcedure=all
```

When activated, the trace prints the following line to the job log after each COBOL invocation:

```
Returned from procedure {name}, duration=1335875 ns, rc=0
```

The reported duration includes the execution time of the COBOL procedure and of the surrounding container code.

Common errors

Table 35. Common errors and suggested solutions when running the COBOL container. This table lists common errors encountered while running the COBOL container, along with suggested solutions. If you encounter a listed error, try the suggested solution.

Error	Suggested solution
<code>com.ibm.websphere.batch.ilc.IContainerException: mkfifo failed with rc=-1</code>	Ensure that the <code>/\${GRID_JOBLOG_ROOT}/joblogs</code> directory exists and is writeable by the WebSphere Application Server SR user ID.
<code>com.ibm.websphere.batch.ilc.IContainerException: LEChildEnvironment.create failed (CEEPIPI function:0x5A rc:0x0000006F rsn:0x5B400002)</code>	Ensure that <code>PIPIENV</code> is on the <code>LIBPATH</code> (<code>server_region_libpath</code>) and is executable by the WebSphere Application Server SR user ID.
<code>java.lang.UnsatisfiedLinkError: LECENV (Not found in java.library.path)</code>	Ensure that <code>libLECENV.so</code> is on the <code>LIBPATH</code> (<code>server_region_libpath</code>) and is executable by the WebSphere Application Server SR user ID.

Generating COBOL call stubs

You can use the COBOL call stub generator to create a Java call stub to invoke a COBOL program. You add the call stub to a Java-based batch application.

Before you begin

Read the COBOL container overview topic.

Install the following products, which are required to use the COBOL call stub generator:

- IBM Rational Application Developer for WebSphere Software, Version 7.5 or later, which provides the J2EE Connector (J2C) tools
- Java Runtime Environment (JRE), Version 1.6.0 or later

To verify that the Java EE Connector (J2C) tools are enabled, start Rational Application Developer, select the Java EE perspective, and select **File > New > Other**. If the **J2C** wizard is available, then you can create a J2C project and the J2C tools are enabled.

If the Rational Application Developer installation does not have the Java EE Connector (J2C) tools, use Installation Manager to modify your Rational Application Developer installation and install the J2C tools:

1. Start Installation Manager.
2. On the Modify Packages page, select **IBM Rational Application Developer for WebSphere Software > J2EE Connector (J2C) tools > Next**.
3. Complete the installation of the J2C tools.

About this task

You can run the COBOL call stub generator from a command line, an Ant task, or the graphical interface of the Rational Application Developer product.

The COBOL call stub generator performs the following steps:

1. Parses the COBOL source program.
2. Generates the Java code for the call stub to invoke the COBOL program.
3. Generates the Java code for the data bindings for the parameter inputs and return value that are used by the COBOL program. The source program is assumed to be a valid COBOL program that has been parsed and compiled by a COBOL compiler.

The Java data binding classes for the COBOL parameters and return value are generated by the J2C data binding tool, which is part of the Java EE Connector tools of the Rational Application Developer product. When you run the COBOL call stub generator from a command line, the call stub generator and the data binding tool run in a headless Eclipse session. The headless Eclipse session is launched as a separate Java process. When you run the COBOL call stub generator from the graphical interface, the call stub generator and the data binding tool run within the active Rational Application Developer session.

Procedure

1. Create a Rational Application Developer project for the generated code.
2. Check the PROCEDURE statement in the COBOL source file.

The PROCEDURE statement must be listed in the COBOL source file, not in a copybook included by the source file. If the call stub generator does not find the PROCEDURE statement in the COBOL source file, the call stub generator fails with an error:

```
com.ibm.ws.batch.cobol.csgen.exceptions.COBOLParserException:  
Unable to find PROCEDURE statement in file COBOL_source_file
```

Also, the PROCEDURE statement must not contain duplicate parameter names in the USING clause. If the USING clause contains a duplicate parameter name, the call stub generator fails with an error:

```
com.ibm.ws.batch.cobol.csgen.exceptions.COBOLParserException:  
Detected duplicate COBOLDataElement: parameter name
```

3. Update the call stub generator configuration file.
See the topic about the call stub generator configuration file.

4. Run the COBOL call stub generator.

You can run the call stub generator in three ways:

From a command line

- a. Create a script that invokes the call stub generator.
- b. Run the script.

```
$ java -jar lib/COBOLCallStubGenerator.jar script_path/Sample01.cb1 -configFile csg.properties -callStubPackage my.pkg
```

See the topic about invoking the call stub generator from a command line.

From an Ant program

- a. Define an Ant task that invokes the call stub generator.
- b. Specify an Ant build file, such as the `CSG.xml` file that is provided with the COBOL call stub generator, to run the call stub generator.
- c. Run the Ant build file.

See the topic about invoking the call stub generator from an Ant task.

From the Rational Application Developer graphical interface

- a. Configure an Ant build under the **Run > External Tools > External Tools Configuration** menu that invokes the call stub generator.
- b. Specify that Ant build file, such as `CSG.xml`, run the call stub generator.
- c. Run the Ant build file.

See the topic about invoking the call stub generator from a graphical interface.

Results

If the call stub generator invocation is successful, the call stub generator creates a Java call stub to invoke a COBOL program.

With a headless Rational Application Developer invocation, an error might result when the call stub generator runs. Running the call stub generator from a command line or Ant task uses a headless invocation. Examine the following logs to troubleshoot an error:

workspace_path/.metadata/.log

The log identifies whether problems with the Eclipse-based tools, such as the headless invocation did not start, caused the error.

user_home/.eclipse/ibm.software.delivery.platform_7.5.0/configuration

Each headless invocation might create log files in the `configuration` directory. If the log files exist, they might provide troubleshooting information on the error.

What to do next

Use the generated Java call stub to invoke a COBOL program.

Creating a call stub generator configuration file

You can create the call stub to identify your Rational Application Developer installation location and to control Java data binding code generation

Before you begin

Create a Rational Application Developer project for the code that is generated by the COBOL call stub generator.

About this task

The call stub generator configuration file contains settings to identify your Rational Application Developer installation location and to control Java data binding code generation. The configuration file location is specified by the `-configFile` setting for command-line invocations or by the `configFile` attribute in the `<csg>` tag for Ant invocations.

Ensure that the `EclipseHome`, `WorkSpace`, and `EclipseProjectName` required settings in the configuration file are correct for your Rational Application Developer installation.

Procedure

1. Open an editor and create a call stub generator configuration file.
The call stub generator configuration file can have any name. Examples for the COBOL call stub generator in this information center use the `csg.properties` file name.
2. Specify COBOL call stub generator settings in the configuration file.
The following table describes the call stub generator settings.

Table 36. Properties to configure COBOL call stub generator. Use these properties to specify the Rational Application Developer installation location and to control Java data binding code generation.

Name	Default value	Required or Optional	Description
EclipseHome	<div style="background-color: #800040; color: white; padding: 2px; display: inline-block;">Linux</div> /opt/IBM/SDP <div style="background-color: #800040; color: white; padding: 2px; display: inline-block;">Windows</div> C:/Program Files/IBM/SDP	Required	Specifies the fully qualified path to the root directory where Rational Application Developer, or Eclipse, is installed in the file system. Note: Use forward slashes (/), or remember to escape the back slashes (\), in all path names.
WorkSpace		Required	Specifies the fully qualified path to the root directory of the Rational Application Developer, or Eclipse, workspace to be used to create the Java data binding class.
EclipseProjectName		Required	Specifies the name of the project in the Rational Application Developer, or Eclipse, workspace that provides the home for the generated Java class. The project must exist before running the call stub generator.
Platform	Win32	Optional	Specifies the target operating system. Supported values include Win32, AIX, and z/OS. The value is case sensitive.
CodePage	ISO-8859-1	Optional	Specifies the code page of the COBOL data.
FloatingPointFormat	IEEE 754	Optional	Specifies the format of floating points. Supported values include IEEE 754 and IBM Hexadecimal.
ExternalDecimalSign	ASCII	Optional	Specifies the external decimal sign. Supported values include ASCII, EBCDIC, and EBCDIC Custom.
Endian	Little	Optional	Specifies the endian of the COBOL data. Supported values include Little and Big.
RemoteEndian	Little	Optional	Specifies the remote endian of the COBOL data. Supported values include Little and Big.
Quote	DOUBLE	Optional	Specifies the quotation format. Supported values include DOUBLE and SINGLE.
Trunc	STD	Optional	Specifies the way fields are truncated during COBOL move or arithmetic operations. Supported values include STD, BIN, and OPT.
Nsymbol	DBCS	Optional	Specifies the way the N symbol is used in literals and PICTURE clauses, indicating whether to use national or DBCS processing. Supported values include DBCS and NATIONAL.
OverwriteExistingClass	true	Optional	Specifies whether the call stub generator overwrites a class with the same name that is already present in the output directory. Supported values include true and false.
GenerationStyle	Default	Optional	Specifies the generation style. Supported values include Default, Preserve case of names, and Shorten names.
Verbose	false	Optional	Specifies whether to set the trace level to debug. Supported values include true and false.

3. Save the configuration file to a location that the Rational Application Developer product can access.

Example

See the following sample `csg.properties` call stub generator configuration file:

```
# Configuration file for COBOLCallStubGenerator
#####
# EclipseHome specifies the installation location of Rational Application Developer.
#
EclipseHome=full_path_to_Eclipse_directory

#####
# Workspace specifies the location of the Rational Application Developer workspace.
# If it does not exist, the COBOLCallStubGenerator creates the workspace.
#
Workspace=full_path_to_workspace_directory

#####
# EclipseProjectName specifies the project in the workspace
# that will receive the generated call stubs and bindings.
#
EclipseProjectName=Eclipse_project_name

#####
# The target operating system. The permitted options are: Win32, AIX and z/OS.
#
# PARAMETER - Required
# Platform=Win32
Platform=z/OS

#####
# The target codepage.
#
# PARAMETER - Required
CodePage=IBM1047

#####
# The floating point format has only two possible values:
# IEEE 754
# IBM Hexadecimal
# The default is IEEE 54
#
# PARAMETER - Required
FloatingPointFormat=IBM Hexadecimal

# PARAMETER - Required
ExternalDecimalSign=EBCDIC

# PARAMETER - Required
Endian=Big

# PARAMETER - Required
RemoteEndian=Little

# PARAMETER - Required
Quote=DOUBLE

# PARAMETER - Required
Trunc=STD

# PARAMETER - Required
Nsymbol=DBCS

#####
# Possible values for overwriteExistingClass are true or false.
#
# PARAMETER - Required
OverwriteExistingClass=true

#####
# Possible values for GenerationStyle are:
# Default, "Preserve case of names" or "Shorten names"
# Be sure to use quotes for values with space characters in them.
```

```

#
# PARAMETER
GenerationStyle=Default

####
# Verbose sets the trace level to "debug".
# Values for Verbose are either true or false.
#
# PARAMETER - Optional
Verbose=false

```

What to do next

Run the COBOL call stub generator. You can run the call stub generator from a command line, an Ant task, or the Rational Application Developer graphical interface.

Specify the fully qualified path to the call stub generator configuration file name when running the COBOL call stub generator. For command-line invocations, use the `-configFile` setting to specify the file name. For Ant and graphical interface invocations, use the `configFile` attribute in the `<csg>` task to specify the file name.

Invoking the call stub generator from a command line

You can run the COBOL call stub generator from a command line.

Before you begin

Create a Rational Application Developer project for the code that is generated by the COBOL call stub generator.

Create a call stub generator configuration file. Ensure that the settings for the `EclipseHome`, `WorkSpace`, and `EclipseProjectName` required properties are correct.

Check the `PROCEDURE` statement for the COBOL source file.

About this task

To run the COBOL call stub generator from a command line, you must run a command that specifies values for the `configFile` and `callStubPackage` required parameters. You also can specify values for optional parameters in the command.

The command that runs the COBOL call stub generator must use the following syntax:

```

java -jar COBOLCallStubGenerator.jar
  {input cobol file}
  -configFile configuration_file_name
  -callStubPackage package_name
  [-callStubClass class_name]
  [-cobolModule COBOL_module]
  [-workSpace path_name]
  [-eclipseProjectName project_name]
  [-build build_string]
  [-mockPackage package_name]
  [-mockClass class_name]
  [-v | -verbose]
  [-? | -h]

```

Table 37. Command parameters to run the COBOL call stub generator. Command parameters have the following definitions.

Name	Default value	Required or Optional	Description
configFile		Required	Specifies the name of the call stub generator configuration file.

Table 37. Command parameters to run the COBOL call stub generator (continued). Command parameters have the following definitions.

Name	Default value	Required or Optional	Description
callStubPackage		Required	Specifies the package name, and the file path of generated file, to use for the generated call stub.
callStubClass	COBOL PROGRAM-ID	Optional	Specifies the class name to use for the generated call stub.
cobolModule	COBOL PROGRAM-ID	Optional	Specifies the name of the COBOL module or DLL file that contains the COBOL program.
workSpace		Optional	Specifies the fully qualified path to the root directory of the Rational Application Developer or Eclipse workspace to be used to create the Java data binding class. Any value that is specified for workSpace overrides the WorkSpace value in the call stub generator configuration file, which is identified by the configFile value.
eclipseProjectName		Optional	Specifies the name of the project in the Rational Application Developer or Eclipse workspace that provides the home for the generated Java class. The project must exist before running the call stub generator. Any value that is specified for eclipseProjectName overrides the EclipseProjectName value in the call stub generator configuration file, which is identified by the configFile value.
build	Unknown Build	Optional	Specifies a user-assigned build identifier.
mockPackage		Optional	Specifies the name of the package for the mock class.
mockClass		Optional	Specifies the name of the class to use for the mock test harness.

Procedure

1. Create a script that runs the call stub generator.
2. Ensure that a Rational Application Developer session is not currently active for the workspace.
3. Run the script.

For example, run the following command to run the `Sample01.cbl` script. The configuration file is named `csg.properties` and the stub package is named `my.pkg`.

```
$ java -jar lib/COBOLCallStubGenerator.jar script_path/Sample01.cbl -configFile csg.properties -callStubPackage my.pkg
```

Results

The command runs the call stub generator. If the command is successful, the call stub generator creates a COBOL call stub Java class.

When running the command, the call stub generator creates a temporary Ant build file, `system_temp_directory/CSG.temp.xml`, and then runs the file in a headless Eclipse session. The temporary file is deleted after the command runs.

The Java data binding classes for the COBOL parameters and return value are generated by the J2C data binding tool, which is part of the Java EE Connector tools of IBM Rational Application Developer for WebSphere Software, Version 7.5 or later. The call stub generator and the J2C data binding tool run in the headless Eclipse session, which is launched as a separate Java process.

The headless invocation of Eclipse attempts to obtain a workspace lock before running the Ant build file. If a Rational Application Developer session is currently active for the workspace, Eclipse cannot obtain a workspace lock and the command fails.

What to do next

Use the generated COBOL call stub Java class.

Invoking the call stub generator from an Ant task

You can run the COBOL call stub generator from an Ant task.

Before you begin

Create a Rational Application Developer project for the code that is generated by the COBOL call stub generator.

Create a call stub generator configuration file. Ensure that the settings for the EclipseHome, WorkSpace, and EclipseProjectName required properties are correct.

Check the PROCEDURE statement for the COBOL source file.

About this task

To run the COBOL call stub generator from an Ant task, you must create a `<csg>` task that specifies values for required attributes that identify the call stub generator configuration file and the COBOL source files that are used as source for the call stub generator. You also can specify values for optional attributes in the task. Table 1 describes supported `<csg>`, `<cobolModule>`, and `<dataElement>` attributes and nested elements that you can use in the `<csg>` task.

In some of the `<csg>` attributes and nested elements, you can specify internal call stub generator properties as substitution variables. For example, for the `<callStubPackage>` element, you can specify the COBOL PROGRAM-ID:

```
<callStubPackage>com.ibm.cobol.$_ProgramId_$</callStubPackage>
```

At run time, the `$_ProgramId_$` variable is substituted with the COBOL PROGRAM-ID. If several COBOL files are specified by the input `<fileset>` or `<filelist>`, then `$_ProgramId_$` is substituted with the PROGRAM-ID of each COBOL file as the call stub generator iterates over the input file list.

Table 38. <csg> attributes and elements to run the COBOL call stub generator. You must specify values for the required attributes and elements in the <csg> task. Many of the optional attributes and elements have a default value. The table first describes <csg> attributes and nested elements, then <cobolModule> attributes and a nested element, and finally <dataElement> attributes.

Name	Default value	Required or Optional	Description
<code><csg></code> attributes and nested elements			
<code>configFile</code>		Required	Specifies the fully qualified path to the call stub generator configuration file.
<code>workSpace</code>		Optional	Specifies the fully qualified path to the root directory of the Rational Application Developer or Eclipse workspace to be used to create the Java data binding class. Any value that is specified for <code>workSpace</code> overrides the <code>WorkSpace</code> value in the call stub generator configuration file, which is identified by the <code>configFile</code> value.
<code>eclipseProjectName</code>		Optional	Specifies the name of the project in the Rational Application Developer or Eclipse workspace that provides the home for the generated Java class. The project must exist before running the call stub generator. Any value that is specified for <code>eclipseProjectName</code> overrides the <code>EclipseProjectName</code> value in the call stub generator configuration file, which is identified by the <code>configFile</code> value.
<code>antBuildFile</code>	<code>\${workSpace}/ \${eclipseProjectName}/src/ GenAllBindings.xml</code>	Optional	Specifies the output location for the data binder Ant build file that is generated by the <code><csg></code> task.
<code><fileset></code> , <code><filelist></code>		Required	Specifies one or more COBOL source files that provide input to the call stub generator. You can specify multiple <code><fileset></code> and <code><filelist></code> types. <code><fileset></code> and <code><filelist></code> types that are specified outside of a <code><cobolModule></code> element use the COBOL PROGRAM-ID as the COBOL module name.

Table 38. <csg> attributes and elements to run the COBOL call stub generator (continued). You must specify values for the required attributes and elements in the <csg> task. Many of the optional attributes and elements have a default value. The table first describes <csg> attributes and nested elements, then <cobolModule> attributes and a nested element, and finally <dataElement> attributes.

Name	Default value	Required or Optional	Description
<cobolModule>		Optional	Specifies the name of the COBOL module or DLL file that is associated with all the <fileset> and <filelist> files that are nested within the <cobolModule> element. You can specify multiple <cobolModule> elements.
<callStubPackage>		Required	Specifies the package name and the file path of generated file to use for the generated call stub. For this element, you can specify an internal call stub generator property for the substitution variables \$_ProgramId_\$, the COBOL PROGRAM-ID, or \$_CobolModule_\$, the COBOL module. For example: <callStubPackage>com.ibm.cobol.\$_ProgramId_\$</callStubPackage> At run time, the \$_ProgramId_\$ variable is substituted with the COBOL PROGRAM-ID.
<callStubClass>	COBOL program PROGRAM-ID	Optional	Specifies the class name to use for the generated call stub. For this element, you can specify an internal call stub generator property for the substitution variables \$_ProgramId_\$, the COBOL PROGRAM-ID, or \$_CobolModule_\$, the COBOL module.
<build>	Unknown Build	Optional	Specifies a user-assigned build identifier.
<mockPackage>		Optional	Specifies the name of the package for the mock class.
<mockClass>		Optional	Specifies the name of the class to use for the mock test harness.
<dataElementsPackage>	{callStubPackage}.parameters	Optional	Specifies the package name to use for the data binding classes of one or more COBOL parameters and the return value. For this element, you can specify an internal call stub generator property for the substitution variables \$_DataElementName_\$, the COBOL parameter name, \$_ProgramId_\$, the COBOL PROGRAM-ID, or \$_CobolModule_\$, the COBOL module.
<dataElementsClass>	Parameter name	Optional	Specifies the class name to use for the data binding classes of one or more COBOL parameters and the return value. For this element, you can specify an internal call stub generator property for the substitution variables \$_DataElementName_\$, the COBOL parameter name, \$_ProgramId_\$, the COBOL PROGRAM-ID, or \$_CobolModule_\$, the COBOL module. For example, to prepend the PROGRAM-ID onto the class names for the COBOL parameters and return value: <dataElementsClass>\$_ProgramId_\$\$_DataElementName_\$</dataElementsClass> At run time, the \$_ProgramId_\$ variable is substituted with the COBOL PROGRAM-ID and the \$_DataElementName_\$ variable is substituted with the name of the COBOL parameter or return value. If the COBOL program has several parameters, then \$_DataElementName_\$ is substituted with the name of each COBOL parameter as the call stub generator iterates over them.
<dataElement>		Optional	Specifies a package and class name for a COBOL parameter or return value. The parameter is identified using the name and programId attributes. The package and class names are specified using the packageName and className attributes. You can specify multiple <dataElement> elements.
<cobolModule> attributes and nested element			
libname	COBOL program PROGRAM-ID	Optional	Specifies the name of the COBOL module or DLL file that contains one or more COBOL programs.
<fileset>, <filelist>		Required	Specifies one or more COBOL source files that provide input to the call stub generator. Each COBOL file is associated with the COBOL module that is defined by libname.
<dataElement> attributes			
name		Required	Specifies the name of the COBOL PROCEDURE parameter.
programId		Required	Specifies the PROGRAM-ID of the COBOL program that contains the parameter.
packageName	{callStubPackage}.parameters	Optional	Specifies the package name to use for the data binding class for this COBOL parameter or return value. For this attribute, you can specify an internal call stub generator property for the substitution variables \$_DataElementName_\$, the COBOL parameter name, \$_ProgramId_\$, the COBOL PROGRAM-ID, or \$_CobolModule_\$, the COBOL module.
className	Parameter name	Optional	Specifies the class name to use for the data binding class for this specific COBOL parameter or return value. For this attribute, you can specify an internal call stub generator property for the substitution variables \$_DataElementName_\$, the COBOL parameter name, \$_ProgramId_\$, the COBOL PROGRAM-ID, or \$_CobolModule_\$, the COBOL module.

Procedure

1. Define an Ant <csg> task to run the call stub generator.

The call stub generator provides a sample Ant build file, CSG.xml, that you can modify and use. See the topic about the call stub generator CSG.xml file.

2. Use an Ant build file to run the call stub generator.

The Ant build file starts the data binder and generates the data binding classes for one or more COBOL parameters and a return value. The data binder Ant build file is run on the last line using the <ant> task.

The following code runs the call stub generator using the <csg> task:

```
<!-- Declare the <csg> task -->
<taskdef resource="com/ibm/ws/batch/cobol/ant/callstub/antlib.xml"
  classpath="{csgDir}/lib/COBOLCallStubGenerator.jar"/>

<csg configFile="{csgDir}/csg.properties" antBuildFile="{antBuildFile}" >

  <!-- Input can be a <fileset> or <filelist>. In this example, it is a single file fileset. -->
  <fileset file="{cobolSource}" />

  <!-- Specify the package for the generated Java call stub class.
  The parameter classes are generated under ${callStubPackage}.parameters. -->
  <callStubPackage>${callStubPackage}</callStubPackage>

</csg>

<!-- Run the data binder Ant build file that is generated by the <csg> task. -->
<ant antfile="{antBuildFile}" />
```

In this example, a <fileset> type is used to specify a single input COBOL source file. COBOL source files used as input to <csg> are specified using one or more <fileset> or <filelist> types. The nested <callStubPackage> element is required. It specifies the package to use for the Java call stub. The data binding classes are generated by default under the \${callStubPackage}.parameters package.

The following example provides a full description of the <csg> task:

```
<taskdef resource="com/ibm/ws/batch/cobol/ant/callstub/antlib.xml"
  classpath="{csgDir}/lib/COBOLCallStubGenerator.jar"/>

<csg configFile="#REQUIRED"
  workspace="#OPTIONAL:overrides value in configFile"
  eclipseProjectName="#OPTIONAL:overrides value in configFile"
  antBuildFile="#OPTIONAL:default=${workspace}/${eclipseProjectName}/src/GenAllBindings.xml" >

  <fileset />
  <filelist />

  <cobolModule #OPTIONAL libname="#OPTIONAL:default=${ProgramId}">
    <fileset />
    <filelist />
  </cobolModule>

  <callStubPackage>#REQUIRED</callStubPackage>
  <callStubClass>#OPTIONAL:default=${ProgramId}</callStubClass>

  <build>#OPTIONAL:default="Unknown Build"</build>
  <mockPackage>#OPTIONAL</mockPackage>
  <mockClass>#OPTIONAL</mockClass>

  <dataElementsPackage>#OPTIONAL:default=${callStubPackage}.parameters</dataElementsPackage>
  <dataElementsClass>#OPTIONAL:default=${dataElementName}</dataElementsClass>

  <dataElement #OPTIONAL
    name="#REQUIRED"
    programId="#REQUIRED"
    packageName="#OPTIONAL:default=${callStubPackage}.parameters"
    className="#OPTIONAL:default=${dataElementName}" />

</csg>
```

Note: The <csg> task and the data binder Ant build file have runtime dependencies on some Eclipse classes, including the Java EE Connector (J2C) tools. Run the <csg> task within Rational

Application Developer as an Ant build to obtain all the prerequisites. You also can use CSG.xml to run the call stub generator within Rational Application Developer. See the topic about invoking the call stub generator using the graphical interface

Results

The command runs the call stub generator. If the command is successful, the call stub generator creates a Java call stub to run a COBOL program.

What to do next

Use the generated Java call stub to run a COBOL program.

Invoking the call stub generator from a graphical interface

You can run the COBOL call stub generator as an Ant build from the Rational Application Developer graphical interface.

Before you begin

Create a Rational Application Developer project for the code that is generated by the COBOL call stub generator.

Create a call stub generator configuration file. Ensure that the settings for the EclipseHome, WorkSpace, and EclipseProjectName required properties are correct.

Check the PROCEDURE statement for the COBOL source file.

About this task

To run the COBOL call stub generator from the Rational Application Developer graphical interface, specify the <csg> task file to use and run the task file in the same Java runtime environment (JRE) as the workspace.

For the <csg> task file, you can use the sample Ant build file, CSG.xml, that is available with the call stub generator of the IBM Rational Application Developer for WebSphere Software product. For CSG.xml input property descriptions, see the topic about the call stub generator CSG.xml file. For descriptions of the <csg> attributes and elements, see the topic about invoking the call stub generator from an Ant task.

The <csg> task and the data binder Ant build file have runtime dependencies on some Eclipse classes, including the Java EE Connector (J2C) tools. When you run the <csg> task within Rational Application Developer as an Ant build, the product obtains all the prerequisites.

Procedure

1. Configure Ant builds to run the <csg> task file.
 - Specify the <csg> task file on the External Tools Configuration page.
 - a. Click **Run > External Tools > External Tools Configurations**.
 - b. For **Buildfile** on the **Main** tab, specify the full path name for the <csg> task file.
For example, to use the CSG.xml Ant build file, specify:
`product_installation_root/CobolCallStubGenerator.V1.2/CSG.xml`
 - c. For **Arguments** on the **Main** tab, specify values for the required CSG.xml input properties.
For example, for the CSG.xml Ant build file, specify values such as the following for required properties:

```
-DcobolSource=${resource_loc}
-Dworkspace=${workspace_loc}
-DclipseProjectName=${project_name}
-DcallStubPackage=${string_prompt:callStubPackage}
```

For property descriptions, including descriptions of the built-in variables such as `resource_loc`, see the topic about the call stub generator `CSG.xml` file.

- d. On the **Main** tab, select **Set an Input handler**
 - e. On the **JRE** tab, select **Run in the same JRE as the workspace**.
This setting ensures that the JRE contains the required Java EE Connector (J2C) tools.
 - f. Optional: On the **Refresh** tab, specify that the Ant build tool build the project or workspace before the tool runs.
 - g. Optional: On the **Build** tab, specify that the Ant build tool refresh the project or workspace after the tool runs.
 - h. Click **Apply** to save the configuration.
2. Optional: Set the copybook include path for the Rational Application Developer workspace.
See the topic about setting COBOL importer preferences in the Rational documentation.
 3. Run the call stub generator.
 - a. On the Package Explorer view, select a COBOL source file.
For example, if your COBOL source file is named `my_source.cb1`, select `my_source.cb1` in the navigation tree of the Package Explorer.
 - b. From the **Run** menu, select **External Tools > 1 CSG Ant** to run the call stub generator.
The first time that you run the tool, the call stub generator Ant file might not show up as an option under the **External Tools** menu. To fix this problem, run the tool from the External Tools Configuration page.

Results

The call stub generator is run and the output is printed to the Rational Application Developer console. The call stub generator creates a COBOL call stub Java class.

What to do next

Use the generated COBOL call stub Java class.

Call stub generator CSG.xml file

The `CSG.xml` file provides a complete example of how to start the COBOL call stub generator from Ant using the `<csg>` custom task or within Rational Application Developer as an Ant build.

CSG.xml input properties

The following table describes the `CSG.xml` input properties. If you run the `<csg>` task in the Rational Application Developer graphical interface, you can define the properties using the `-Dproperty_name=property_value` format.

Table 39. CSG.xml input properties. Specify values for these properties in the CSG.xml file.

Name	Required or Optional	Description
cobolSource	Required	Specifies the fully qualified path to the COBOL source file. Note: Use forward slashes (/), or remember to escape the back slashes (\), in all path names. In the Rational Application Developer graphical interface, you can use the <code>\${resource_loc}</code> built-in variable, which is substituted with the fully qualified path of the resource currently selected on the Package Explorer.

Table 39. CSG.xml input properties (continued). Specify values for these properties in the CSG.xml file.

Name	Required or Optional	Description
workSpace	Required	Specifies the fully qualified path to the root directory of the Rational Application Developer or Eclipse workspace to be used to create the Java data binding class. In the Rational Application Developer graphical interface, you can use the <code>\${workspace_loc}</code> built-in variable, which is substituted with the fully qualified path of the current active workspace.
eclipseProjectName	Required	Specifies the name of the project in the Rational Application Developer or Eclipse workspace that provides the home for the generated Java class. In the Rational Application Developer graphical interface, you can use the <code>\${project_name}</code> built-in variable, which is substituted with the project name of the resource currently selected on the Package Explorer. The project must exist before running the call stub generator.
callStubPackage	Required	Specifies the package name to use for the generated call stub. In the Rational Application Developer graphical interface, you can use the <code>\${string_prompt:callStubPackage}</code> built-in variable. At run time, Rational Application Developer prompts you for a string value. The title of the prompt dialog box is callStubPackage .

File location

The CSG.xml Ant build file has a location such as:

```
product_installation_root/CobolCallStubGenerator.V1.2/CSG.xml
```

CSG.xml contents

The CSG.xml file that is provided with the product resembles the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ..... -->
<!-- This Ant build file is used for invoking the COBOLCallStubGenerator (CSG)
within Rational Application Developer. It has two steps:
1. Calls CSG for the given cobolSource file.
   CSG is invoked using the custom <csg> task.
   <csg> generates the Java call stub and the data binder Ant file.
2. Invokes the data binder Ant file that is generated by <csg>.
   The data binder Ant file invokes the data binder and generates
   the data binding Java classes for the COBOL parameters.

To configure this file for use within Rational Application Developer:
1. Run -> External Tools -> External Tools Configuration.
2. New Ant Build
   Buildfile: full_path_to_this_file.xml
   Arguments: -DcobolSource=${resource_loc}
              -DworkSpace=${workspace_loc}
              -DeclipseProjectName=${project_name}
              -DcallStubPackage=${string_prompt:callStubPackage}
   Refresh: project
   Build: project
   JRE: same as workspace (needs J2C tools installed)

Usage instructions:
1. Load the COBOL source into the project.
2. Select the COBOL source file or source directory in Project Explorer.
3. Click Run -> External Tools -> CSG Ant build.
-->
<!-- ..... -->

<project default="CSG">
  <property name="csgDir" value="${basedir}" />
  <property name="debug" value="false" />
```

```

<!-- Include the <csg> and <radlogcheck> task definitions -->
<taskdef resource="com/ibm/ws/batch/cobol/ant/callstub/antlib.xml"
    classpath="${csgDir}/lib/COBOLCallStubGenerator.jar"/>

<!-- These input properties are required. They can be specified as arguments to the Ant build. -->
<property name="cobolSource" value="" /> <!-- full path to COBOL source file -->
<property name="workSpace" value="" /> <!-- full path to Eclipse workspace -->
<property name="eclipseProjectName" value="" />
<property name="callStubPackage" value="" />

<!-- The output from the <csg> task is the data binder Ant build file. The data binder Ant
    build file invokes the data binder and generates the data binding classes for all
    the COBOL parameters for all the COBOL files processed by <csg>. -->
<property name="antBuildFile" value="${workSpace}/${eclipseProjectName}/src/GenAllBindings.xml" />

<!-- Quick way to get the directory of the input COBOL file, in case you wanted to use it
    to specify a list of files in the <fileset> type. -->
<dirname property="cobolSourceDir" file="${cobolSource}" />

<target name="CSG">

    <csg configFile="${csgDir}/csg.properties"
        workSpace="${workSpace}"
        eclipseProjectName="${eclipseProjectName}"
        antBuildFile="${antBuildFile}" >

        <!-- In this example, the input is a single COBOL file, specified using a <fileset>.
            You can modify the input to be a list of COBOL files, using either the
            <fileset> or <filelist> types. For example, instead of using the cobolSource
            property, you can use the cobolSourceDir property to specify all *.cbl files
            in the directory or use <filelist> to list a subset of files in the directory.
            Examples for setting the input to a list of files are included here, but commented out. -->
        <fileset file="${cobolSource}" />
        <!-- Example: Set the input to all *.cbl files in a directory: -->
        <!-- <fileset dir="${cobolSourceDir}" includes="**/*.cbl" /> -->
        <!-- Example: Set the input to a subset of files in a directory: -->
        <!-- <filelist dir="${cobolSourceDir}" files="G10M0802.cbl primitve.ccp natltest.ccp" /> -->
        <!-- Example: Associate all the nested files with the specified <cobolModule>: -->
        <!-- <cobolModule libname="MyDLLName">
            <fileset dir="${cobolSourceDir}" includes="**/*.ccp" />
        </cobolModule> -->
        <!-- You can specify multiple <cobolModule> elements. -->

        <callStubPackage>${callStubPackage}</callStubPackage>
        <!-- Example: You can use some internal call stub generator properties as substitution
            variables. For example, to include the COBOL PROGRAM-ID (${_ProgramId_$}) in the
            callStubPackage: -->
        <!-- <callStubPackage>${callStubPackage}.${_ProgramId_$}</callStubPackage> -->

        <!-- If not defined, the package name for the data binding classes is
            ${callStubPackage}.parameters. -->
        <!-- <dataElementsPackage>${callStubPackage}.parameters</dataElementsPackage> -->

        <!-- Example: Use substitution variables to prepend the COBOL PROGRAM-ID to the
            parameter class name. -->
        <!-- <dataElementsClass>_${ProgramId}_${_DataElementName_$}</dataElementsClass> -->

        <!-- The Java package and class for the parameter identified by 'name' in COBOL
            program 'programId' -->
        <!-- <dataElement name="#REQUIRED" programId="#REQUIRED" packageName="" className="" /> -->
        <!-- You can specify multiple <dataElement> elements. -->

    </csg>

    <!-- Refresh the workspace in order to compile the Java call stubs generated by <csg> -->
    <eclipse.refreshLocal depth="infinite" resource="${eclipseProjectName}" />

    <!-- radlogcheck first takes a snapshot of the Rational Application Developer log
        (${workSpace}/.metadata/.log). Later, radlogcheck examines the log and searches
        for any errors generated during this data binding step. -->
    <radlogcheck workSpace="${workSpace}" stage="begin" />

```



```

<!-- Now run the data binder Ant build file that is generated by the <csg> task. -->
<ant antfile="\${antBuildFile}" />

<!-- Scan the log for any errors that occurred during the data binding step.
     If errors are detected, the product writes them to the console. -->
<radlogcheck workspace="\${workspace}" stage="end" />

<!-- Delete the antBuildFile. It is no longer needed and is merely a build artifact. -->
<delete file="\${antBuildFile}" failonerror="false"/>

</target>
</project>

```

Call stub generator CSGBatch.xml file

The CSGBatch.xml Ant build file provides an example for running <csg> against a batch of COBOL files, such as all *.cbl files in a directory. The call stub generator and CSGBatch.xml file are available with the IBM Rational Application Developer for WebSphere Software product.

CSGBatch.xml input properties

CSGBatch.xml uses the same input properties as the CSG.xml file. For CSG.xml input property descriptions, see the topic about the call stub generator CSG.xml file. For descriptions of the <csg> attributes and elements, see the topic about invoking the call stub generator from an Ant task.

If you run the <csg> task in the Rational Application Developer graphical interface, you can define the properties using the *-Dproperty_name=property_value* format.

CSGBatch.xml uses the `$_ProgramId_` and `$_DataElementName_` substitution variables to manipulate the package name and parameter names generated for each COBOL file.

File location

The CSGBatch.xml Ant build file has a location such as:

```
product_installation_root/CobolCallStubGenerator.V1.2/CSGBatch.xml
```

CSGBatch.xml contents

The CSGBatch.xml file that is provided with the product resembles the following example:

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- ..... -->
<!-- This Ant build file can run the COBOLCallStubGenerator (CSG) within Rational
     Application Developer. See CSG.xml for a more complete description of this file.
     Except this file takes a directory of COBOL files (*.cbl) and runs <csg> against
     each file.

     Note: This file is configured to append the COBOL PROGRAM-ID to the callStubPackage
     and to prepend the PROGRAM-ID to each parameter name (see the <callStubPackage>
     and <dataElementsClass> elements).
     ..... -->
<!-- ..... -->

<project default="CSGBatch">
  <property name="csgDir" value="\${basedir}" />
  <property name="debug" value="false" />
<taskdef resource="com/ibm/ws/batch/cobol/ant/callstub/antlib.xml"
  classpath="\${csgDir}/lib/COBOLCallStubGenerator.jar"/>

  <!-- These input properties are required. They can be specified as arguments to the Ant build. -->
  <property name="cobolSourceDir" value="" /> <!-- full path to COBOL source directory -->
  <property name="workspace" value="" /> <!-- full path to Eclipse workspace -->
  <property name="eclipseProjectName" value="" />
  <property name="callStubPackage" value="" />
  <property name="antBuildFile" value="\${workspace}/\${eclipseProjectName}/src/GenAllBindings.xml" />

```



```

<target name="CSGBatch">
  <csg configFile="{csgDir}/csg.properties"
    workspace="{workspace}"
    eclipseProjectName="{eclipseProjectName}"
    antBuildFile="{antBuildFile}" >

    <!-- Process all *.cbl files in the given cobolSourceDir. -->
    <fileset dir="{cobolSourceDir}" includes="**/*.cbl" />

    <!-- Append the COBOL PROGRAM-ID to the callStubPackage using substitution variables. -->
    <callStubPackage>${callStubPackage}._ProgramId_</callStubPackage>

    <!-- Prepend the COBOL PROGRAM-ID to the parameter class name using substitution variables. -->
    <dataElementsClass>$_ProgramId_$_DataElementName_</dataElementsClass>

  </csg>

  <eclipse.refreshLocal depth="infinite" resource="{eclipseProjectName}" />

  <!-- Run the data binder in a radlogcheck to detect and print errors. -->
  <radlogcheck workspace="{workspace}" stage="begin" />
  <ant antfile="{antBuildFile}" />
  <radlogcheck workspace="{workspace}" stage="end" />

  <!-- Delete the antBuildFile. It is no longer needed and is merely a build artifact. -->
  <delete file="{antBuildFile}" failonerror="false"/>
</target>
</project>

```

Developing a simple compute-intensive application

You can write a simple compute-intensive application using a compute-intensive job controller, the command line, or the Apache ANT tool.

Procedure

- Create a compute-intensive job using a compute-intensive job controller.

1. Create a compute-intensive job step.

- a. Create a Java class that implements the `com.ibm.websphere.ci.CIWork` interface.
- b. Implement business logic.

2. Optional: For batch applications, provide a job listener.

Provide an implementation for the `com.ibm.websphere.batch.listener.JobListener` interface to add additional initialization and clean up for jobs and steps. Specify the job listener in the xJCL using the job-level listener element.

The job listener `beforeJob()` method is invoked before any user artifact is invoked. The job listener `afterJob()` method is invoked after the last user artifact is invoked. The job listener `beforeStep()` method is invoked before any step-related user artifact. The job listener `afterStep()` method is invoked as the last step-related user artifact. Each time the job listener is invoked, it logs a message to the job log.

3. Optional: For batch applications, obtain the job step context.

```
JobStepContext ctx= JobStepContextMgr.getContext();
```

The `JobStepContextMgr` service class enables the batch job step to obtain a reference to its `JobStepContext` object. The job step context provides the following capabilities:

- Access to information that uniquely identifies the context in which the current batch job step runs, for example, the job ID
- A user data area where application-specific information can be passed among the batch programming framework methods during the life of the batch job step
- A transient user data area where application-specific information can be passed across steps

- A persistent user data area where application-specific information is stored across checkpoint/restart

You can use the PersistentMap helper class to simplify the storing of basic types such as boolean and double in the persistent user data area of the job step context.

4. Declare a compute-intensive job controller.
 - a. Add a stateless session bean to your deployment descriptor and point to the implementation class that the product provides.
Do so by specifying `com.ibm.ws.ci.CIControllerBean` as the bean class. Do this specification only once per compute-intensive application.
 - b. Use `com.ibm.ws.ci.CIControllerHome` for the remote home interface class and `com.ibm.ws.ci.CIController` for the remote interface class.
5. Configure the EJB deployment descriptor.
 - a. Configure a resource reference on the controller bean to the `WorkManager` `wm/CIWorkManager` default of the type `commonj.work.WorkManager`.

- Create compute-intensive jobs using the command line.

1. Create a compute-intensive job step.
 - a. Create a Java class that implements the `com.ibm.websphere.ci.CIWork` interface.
 - b. Implement business logic.
2. Optional: For batch applications, obtain the job step context.

```
JobStepContext ctx= JobStepContextMgr.getContext();
```

The `JobStepContextMgr` service class enables the batch job step to obtain a reference to its `JobStepContext` object. The job step context provides the following capabilities:

- Access to information that uniquely identifies the context in which the current batch job step runs, for example, the job ID
- A user data area where application-specific information can be passed among the batch programming framework methods during the life of the batch job step
- A transient user data area where application-specific information can be passed across steps
- A persistent user data area where application-specific information is stored across checkpoint/restart

You can use the PersistentMap helper class to simplify the storing of basic types such as boolean and double in the persistent user data area of the job step context.

3. Open a command prompt and ensure that the directory where your Java executable program is located is included in your PATH variable so that you can run the Java command.
4. Issue a Java command.

```
java -jar pgcbatchpackager.jar -appname=<application name>
-jarfile=<jarfile containing POJO step classes> -earfile=<name of the output ear file without .ear>
[-utilityjars=<semicolon separated list of utility jars>] [-debug] [-gridjob]
```

For example for batch jobs, issue the command:

```
java -jar pgcbatchpackager.jar -appname=SimpleCI -jarfile=SimpleCIEJBs.jar
-earfile=SimpleCI -gridjob=true
```

- Create compute-intensive jobs using ANT.

1. Create the compute-intensive job step.
 - a. Create a Java class that implements the `com.ibm.websphere.ci.CIWork` interface.
 - b. Implement business logic.
2. Optional: For batch applications, obtain the job step context.

```
JobStepContext ctx= JobStepContextMgr.getContext();
```

The `JobStepContextMgr` service class enables the batch job step to obtain a reference to its `JobStepContext` object. The job step context provides the following capabilities:

- Access to information that uniquely identifies the context in which the current batch job step runs, for example, the job ID
- A user data area where application-specific information can be passed among the batch programming framework methods during the life of the batch job step
- A transient user data area where application-specific information can be passed across steps
- A persistent user data area where application-specific information is stored across checkpoint/restart

You can use the PersistentMap helper class to simplify the storing of basic types such as boolean and double in the persistent user data area of the job step context.

3. For a compute-intensive job, ensure that `pgcbatchpackager.jar` is on the class path.
4. Declare the task.

Use the following command to declare the task:

```
<taskdef name="pgcpackager" classname="com.ibm.ws.batch.packager.PGCPackager"
        classpath="${FEBaseDir}/grid.pgc.packager/build/lib/pgcbatchpackager.jar" />
```

5. After compiling the Java files in your application, invoke the `pgcpackager` task.

```
<pgcpackager appname="<appname>" earFile="<location name of EAR file to generate>"
jarfile="location of the POJO jar file" gridJob="true"/>
```

Results

You have developed a simple compute-intensive application using a compute-intensive job controller, the command line, or ANT.

What to do next

Install the compute-intensive application and configure WebSphere grid endpoints.

Compute-intensive programming model

Compute-intensive applications are applications that perform computationally intensive work that does not fit comfortably into the traditional Java Platform, Enterprise Edition (Java EE) request and response paradigm.

Compute-intensive applications

There are a number of characteristics that can make these applications unsuitable for traditional Java EE programming models:

- The need for asynchronous submission and start of work
- The need for work to run for extended periods of time
- The need for individual units of work to be visible to and manageable by operators and administrators

The compute-intensive programming model provides an environment that addresses these needs. The compute-intensive programming model is centered around two basic concepts:

1. The use of jobs to submit and manage work asynchronously
2. A minor extension to the asynchronous beans programming model to support work that runs for an extended period

The following sections provide additional information about the extensions to the asynchronous beans programming model.

Controller bean

The controller bean is a stateless session bean defined in the compute-intensive application deployment descriptor that allows the runtime environment to control jobs for the application. The implementation of

this stateless session bean is provided by the application server. The application includes the stateless session bean, shown in the following definition, in the deployment descriptor of one of its enterprise bean modules. Exactly one controller bean must be defined for each compute-intensive application. Since the implementation of the controller bean is provided in the application server runtime, application deployers do not request deployment of enterprise beans during deployment of compute-intensive applications.

```
<session id="supply a suitable name here">
  <ejb-name>CIController</ejb-name>
  <home>com.ibm.ws.ci.CIControllerHome</home>
  <remote>com.ibm.ws.ci.CIController</remote>
  <ejb-class>com.ibm.ws.ci.CIControllerBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Bean</transaction-type>
  <resource-ref id="WorkManager_ResourceRef">
    <description>
      WorkManager that is used to execute jobs.
    </description>
    <res-ref-name>wm/CIWorkManager</res-ref-name>
    <res-type>commonj.work.WorkManager</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
</session>
```

Packaging a compute-intensive application

The logic for a compute-intensive application with some number of CIWork objects plus the classes to support those CIWork objects, is packaged in an enterprise bean module in a Java EE application Enterprise Archive (EAR) file. The deployment descriptor for the enterprise bean module must contain the definition of the stateless session bean previously described. If the application itself uses other enterprise beans or resources, then the definitions for those beans and resources might also be in the deployment descriptor. You can use Java EE development tools such as IBM Rational Application Developer to develop and package compute-intensive applications in the same way that they are used to construct Java EE applications containing enterprise bean modules and asynchronous beans. You can also use the `pgcpackager` task to package compute-intensive applications.

Life cycle of a compute-intensive application

A compute-intensive application is started by the application server in the same way as other Java EE applications. If the application defines any start-up beans, then those beans are run when the application server starts. When a job arrives for the application to run, the compute-intensive execution environment invokes the `CIControllerBean` stateless session bean defined in the application EJB module deployment descriptor. The Java Naming and Directory Interface (JNDI) name of this stateless session bean is specified in the XML Job Control Language (xJCL) for the job. For each job step, the `CIControllerBean` stateless session bean completes the following actions:

1. Instantiates the application CIWork object specified by the class name element in the xJCL for the job step using the no-argument constructor of the CIWork class.
2. Invokes the `setProperties()` method of the CIWork object to pass any properties defined in the xJCL for the job step.
3. Looks up the work manager defined in the deployment descriptor of the enterprise bean module and uses it to asynchronously call the `run()` method of the CIWork object.

If the job is canceled before the `run()` method returns, then the `CIControllerBean` invokes the `CIWork` object `release()` method on a separate thread. It is up to the developer of the long-running application to arrange for logic in the `release()` method to cause the `run()` method to return promptly. The job remains in a *cancel pending* state until the `run()` method returns.

If the job is not canceled and the `run()` method returns without returning an exception, then the job completed successfully. If the `run()` method returns an exception, then the job status is *execution failed*. After the `run()` method returns either successfully or with an exception, no further calls are made to the CIWork object. All references to the `run()` method are dropped.

Compute-intensive job step

Unlike other batch jobs, compute-intensive jobs consist of one job step. This job step is represented by an instance of a class that implements the `com.ibm.websphere.ci.CIWork` interface. The `CIWork` interface extends the `commonj.Work` interface from the application server asynchronous beans programming model and Java Specification Request (JSR) 237. These extensions consist of two methods that provide a way to pass the job-step-specific properties specified in the job to the `CIWork` object.

See the API documentation for more details.

To learn about asynchronous beans, see [Using asynchronous beans](#).

Developing a simple transactional batch application

You can write a simple batch application using a batch job controller and Enterprise JavaBeans (EJB) data stream, the command line, or the Apache ANT tool.

About this task

Note: If the batch step uses a batch data stream (BDS) whose data is local to the file system of the application server to which the batch application is deployed, then certain steps must be followed to support job restart scenarios. If such a batch application is deployed to application servers that can run on multiple machines, then there is no guarantee that the restart request is accepted by the machine on which the batch job originally ran. This occurs when the batch application is deployed to a cluster, and if a batch job that runs against such an application is canceled and then restarted. In this scenario, the deployment might send the restart request to an application server that runs on a different machine. Therefore, in cases where file-based affinity is required, you can apply the following solutions to support the job restart scenario:

- Ensure that the data is equally available to every machine on which the batch application can be started. Use a network file system for this example. This action might reduce performance of application.
- Deploy the application on application servers that can only run on the machine where the local data exists. Complete this action by deploying the application to a cluster that exists in a node group that has only one member node.

Note: The batch application developer must ensure that transactional work done in the batch step callback methods inherits the global transaction started by the grid endpoints. This action ensures that work performed under a batch step only gets committed at every checkpoint and rolls back if the step fails.

Some commands are split on multiple lines for printing purposes.

Procedure

- Create batch jobs using a batch job controller and an EJB data stream.
 1. Create batch job steps.
 - a. Create a Java class that implements the `com.ibm.websphere.BatchJobStepInterface` interface.
 - b. Implement business logic.

If your step has one input and one output stream you can alternatively use the Generic batch step of `GenericXDBatchStep`.
 2. Create batch data streams.
 - a. Create a Java class that implements the interface `com.ibm.websphere.batch.BatchDataStream`.

Batch data streams are accessed from the business logic, for example, from the batch job steps by calling `BatchDataStreamMgr` with `jobID` and `stepID`. `JobID` and `stepID` are retrieved from the step bean properties list using keys `BatchConstants.JOB_ID` and `BatchConstants.STEP_ID`.

- b. Map `BatchConstants.JOB_ID` to `com.ibm.websphere.batch.JobID` and map `BatchConstants.STEP_ID` to `com.ibm.websphere.batch.StepID`.

You should already have access to the `BatchConstants` class.

The batch datastream framework provides several ready-to-use patterns to work with different types of datastreams such as file and database. To use the batch datastream framework, complete the following steps.

- 1) Identify the data stream type with which you want to operate, such as `TextFile`, `ByteFile`, `JDBC`, or `z/OS` stream.
- 2) Identify whether you would read from the stream or write to the stream.
- 3) See the table in the batch data stream framework and patterns. Select the class from the supporting classes column that matches your data stream type and operation. For example, if you want to read data from a text file, then select `TextFileReader`.
- 4) Implement the interface listed in the pattern name column that corresponds to the supporting class you selected in the previous step. The supporting class handles all the book keeping activities related to the stream and the batch programming model. The implementation class focuses on the stream processing logic.
- 5) Declare the supporting class and your implementation class in the xJCL.
- 6) Repeat this procedure for each datastream required in your step.

3. Optional: Obtain the job step context.

```
JobStepContext ctx= JobStepContextMgr.getContext();
```

The `JobStepContextMgr` service class enables the batch job step to obtain a reference to its `JobStepContext` object. The job step context provides the following capabilities:

- Access to information that uniquely identifies the context in which the current batch job step runs, for example, the job ID
- A transient user data area where application-specific information can be passed among the batch programming framework methods during the life of the batch job step
- A persistent user data area where application-specific information can be passed across steps

You can use the `PersistentMap` helper class to simplify the storing of basic types such as `Boolean` and `double` in the persistent user data area of the job step context.

4. Define batch data streams in xJCL.

```
<batch-data-streams>
  <bds>
    <logical-name>inputStream</logical-name>
    <props>
      <prop name="PATTERN_IMPL_CLASS" value="MyBDSStreamImplementationClass"/>
    <prop name="file.encoding" value="8859_1"/>
    <prop name="FILENAME" value="{inputDataStream}" />
    <prop name="PROCESS_HEADER" value="true"/>
    <prop name="AppendJobIdToFileName" value="true"/> </props>
    <impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteReader
  </impl-class>
</bds>
```

The `PATTERN_IMPL_CLASS` class denotes the user implementation of the BDS framework pattern and the `impl-class` property denotes the supporting class.

5. Optional: Enable skip-level processing.

Use skip-record processing to skip read and write record errors in transactional batch jobs. Specify skip-record policies in the xJCL. Read the topic on skip record processing for further information.

6. Optional: Enable retry-step processing.

Use retry-step processing to try job steps again when the processJobStep method encounters errors in a transactional batch job. Specify retry-step policies in the xJCL. Read the topic on retry-step processing for further information.

7. Optional: Configure the transaction mode.

Use the transaction mode to define whether job-related artifacts are called in global transaction mode or local transaction mode. Read the topic on the configurable transaction mode for further information.

8. Optional: Provide a job listener.

Provide an implementation for the `com.ibm.websphere.batch.listener.JobListener` interface to add additional initialization and clean up for jobs and steps. Specify the job listener in the xJCL using the job-level listener element.

The job listener `beforeJob()` method is invoked before any user artifact is invoked. The job listener `afterJob()` method is invoked after the last user artifact is invoked. The job listener `beforeStep()` method is invoked before any step-related user artifact. The job listener `afterStep()` method is invoked as the last step-related user artifact. Each time the job listener is invoked, it logs a message to the job log.

9. Declare a batch job controller.

a. Add a stateless session bean to your deployment descriptor and point to the implementation class that the product provides. Do so by specifying `com.ibm.ws.batch.BatchJobControllerBean` as the bean class. Do this specification only once per batch application.

b. Use `com.ibm.ws.batch.BatchJobControllerHome` for the remote home interface class and `com.ibm.ws.batch.BatchJobController` for the remote interface class.

10. Configure the EJB deployment descriptor.

a. Configure a resource reference on the Controller bean to the default `WorkManager` `wm/BatchWorkManager` of the type `commonj.work.WorkManager`.

Note: You must declare the deployment descriptor of the batch controller bean in the Enterprise JavaBeans (EJB) deployment descriptor of a batch application. Only one controller bean can be defined per batch application.

• Create batch jobs using the command line.

1. Create batch job steps.

- a. Create a Java class that implements the `com.ibm.websphere.BatchJobStepInterface` interface.
- b. Implement business logic.

If your step has exactly one input and one output stream you could alternatively use the Generic batch step of `GenericXDBatchStep`.

2. Create batch data streams.

- a. Create a Java class that implements the interface `com.ibm.websphere.batch.BatchDataStream`. Batch data streams are accessed from the business logic, for example, from the batch job steps by calling `BatchDataStreamMgr` with `jobID` and `stepID`. `jobID` and `stepID` are retrieved from the step bean properties list using keys `BatchConstants.JOB_ID` and `BatchConstants.STEP_ID`.

- b. Map `BatchConstants.JOB_ID` to `com.ibm.websphere.batch.JobID` and map `BatchConstants.STEP_ID` to `com.ibm.websphere.batch.StepID`.

You should already have access to the `BatchConstants` class.

The batch datastream framework provides several ready-to-use patterns to work with different types of datastreams such as file and database. To use the batch datastream framework, complete the following steps.

- 1) Identify the data stream type with which you want to operate, such as `TextFile`, `ByteFile`, `JDBC`, or `z/OS stream`.
- 2) Identify whether you would read from the stream or write to the stream.

- 3) See the table in the batch data stream framework and patterns. Select the class from the supporting classes column that matches your data stream type and operation. For example, if you want to read data from a text file, then choose TextFileReader.
 - 4) Implement the interface listed in the pattern name column that corresponds to the supporting class you selected in the previous step. The supporting class handles all the book keeping activities related to the stream and the batch programming model. The implementation class focuses on the stream processing logic.
 - 5) Declare the supporting class and your implementation class in the xJCL.
 - 6) Repeat this procedure for each datastream required in your step.
3. Optional: Obtain the job step context.

```
JobStepContext ctx= JobStepContextMgr.getContext();
```

The JobStepContextMgr service class enables the batch job step to obtain a reference to its JobStepContext object. The job step context provides the following capabilities:

- Access to information that uniquely identifies the context in which the current batch job step runs, for example, the job ID
- A transient user data area where application-specific information can be passed among the batch programming framework methods during the life of the batch job step
- A persistent user data area where application-specific information can be passed across steps

You can use the PersistentMap helper class to simplify the storing of basic types such as Boolean and double in the persistent user data area of the job step context.

4. Open a command prompt and ensure that Java is on the path.
5. Issue the following command, all on a single line.

Distributed operating systems

IBM i

```
java -cp ${WAS_INSTALL_ROOT}/plugins/com.ibm.ws.batch.runtime.jar
com.ibm.ws.batch.packager.WSBatchPackager
  -appname=<Application_Name>
  -jarfile=<jarfile containing the POJO batch steps>
  -earfile=<name of the output EAR file without the .ear extension>
  [-utilityjars=<semicolon separated list of utility jars>]
  [-debug]
  [-gridJob]
```

For example, for batch jobs, issue

```
java -cp ${WAS_INSTALL_ROOT}/plugins/com.ibm.ws.batch.runtime.jar
com.ibm.ws.batch.packager.WSBatchPackager
  -appname=XDCGIVT
  -jarfile=XDCGIVTEJBs.jar
  -earfile=XDCGIVT
```

z/OS

```
java -cp ${WAS_INSTALL_ROOT}/plugins/com.ibm.ws.batch.runtime.jar
com.ibm.ws.batch.packager.WSBatchPackager
  -Dfile.encoding=ISO8859-1
  -appname=<Application_Name>
  -jarfile=<jarfile containing the POJO batch steps>
  -earfile=<name of the output EAR file without the .ear extension>
  [-utilityjars=<semicolon separated list of utility jars>]
  [-debug]
  [-gridJob]
```

For example for batch jobs, issue

```
java -cp ${WAS_INSTALL_ROOT}/plugins/com.ibm.ws.batch.runtime.jar
com.ibm.ws.batch.packager.WSBatchPackager
  -Dfile.encoding=ISO8859-1
  -appname=XDCGIVT
  -jarfile=XDCGIVTEJBs.jar
  -earfile=XDCGIVT
```

Note: If you do not include `-Dfile.encoding=ISO8859-1`, code page differences result that yield invalid EAR and Enterprise JavaBeans (EJB) Java archive (JAR) descriptors.

6. Package a batch application.

Use one of the following methods.

- Package the application using the WSBatchPackager script.

Distributed operating systems

```
<WASHOME>/stack_products/WCG/bin/WSBatchPackager.sh
-appname=<application_name>
-jarfile=<jar_file_containing_POJO_step_classes>
-earfile=<output_ear_file_name>
[-utilityjars=<semicolon_separated_utility_jars>]
[-nonxadsjndiname=<non-xa_datasource_JNDI_name_for_CursorHoldableJDBCReader>;<non-XA_datasource_JNDI_name_2>;...]
[-debug]
```

For example, issue

```
./WSBatchPackager.sh -appname=XDCGIVT -jarfile=XDCGIVTEJBs.jar -earfile=XDCGIVT -utilityjars=myutility.jar -nonxadsjndiname=jdbc/ivtnonxa
```

z/OS

```
<WASHOME>/stack_products/WCG/bin/WSBatchPackager.sh
-Dfile.encoding=ISO8859-1
-appname=<application_name>
-jarfile=<jar_file_containing_POJO_step_classes>
-earfile=<output_ear_file_name>
[-utilityjars=<semicolon_separated_utility_jars>]
[-nonxadsjndiname=<non-xa_datasource_JNDI_name_for_CursorHoldableJDBCReader>;<non-XA_datasource_JNDI_name_2>;...]
[-debug]
```

For example, issue

```
./WSBatchPackager.sh -Dfile.encoding=ISO8859-1 -appname=XDCGIVT -jarfile=XDCGIVTEJBs.jar -earfile=XDCGIVT -utilityjars=myutility.jar
-nonxadsjndiname=jdbc/ivtnonxa
```

Note: If you do not include `-Dfile.encoding=ISO8859-1`, code page differences result that yield invalid EAR and Enterprise JavaBeans (EJB) Java archive (JAR) descriptors.

- Package the application using the **java** command.

Open a command prompt and ensure that java is on the path.

- Create batch jobs using ANT.

1. Create batch job steps.

- a. Create a Java class that implements the `com.ibm.websphere.BatchJobStepInterface` interface.
- b. Implement business logic.

If your step has exactly one input and one output stream you could alternatively use the Generic batch step of `GenericXDBatchStep`.

2. Create batch data streams.

- a. Create a Java class that implements the interface `com.ibm.websphere.batch.BatchDataStream`.
Batch data streams are accessed from the business logic, for example, from the batch job steps by calling `BatchDataStreamMgr` with `jobID` and `stepID`. `jobID` and `stepID` are retrieved from the step bean properties list using keys `BatchConstants.JOB_ID` and `BatchConstants.STEP_ID`.
- b. Map `BatchConstants.JOB_ID` to `com.ibm.websphere.batch.JobID` and map `BatchConstants.STEP_ID` to `com.ibm.websphere.batch.StepID`.

You should already have access to the `BatchConstants` class.

The batch datastream framework provides several ready-to-use patterns to work with different types of datastreams such as file and database. To use the batch datastream framework, complete the following steps.

- 1) Identify the data stream type with which you want to operate, such as `TextFile`, `ByteFile`, `JDBC`, or `z/OS` stream.
- 2) Identify whether you would read from the stream or write to the stream.
- 3) See the table in the batch data stream framework and patterns. Select the class from the supporting classes column that matches your data stream type and operation. For example, if you want to read data from a text file, then select `TextFileReader`.

- 4) Implement the interface listed in the pattern name column that corresponds to the supporting class you selected in the previous step. The supporting class handles all the book keeping activities related to the stream and the batch programming model. The implementation class focuses on the stream processing logic.
- 5) Declare the supporting class and your implementation class in the xJCL.
- 6) Repeat this procedure for each datastream required in your step.

3. Optional: Obtain the job step context.

```
JobStepContext ctx= JobStepContextMgr.getContext();
```

The JobStepContextMgr service class enables the batch job step to obtain a reference to its JobStepContext object. The job step context provides the following capabilities:

- Access to information that uniquely identifies the context in which the current batch job step runs, for example, the job ID
 - A transient user data area where application-specific information can be passed among the batch programming framework methods during the life of the batch job step
 - A persistent user data area where application-specific information can be passed across steps
- You can use the PersistentMap helper class to simplify the storing of basic types such as Boolean and double in the persistent user data area of the job step context.

4. For a batch job, ensure the com.ibm.ws.batch.runtime.jar file is on the class path.

5. Declare the task.

Use the following command to declare the task:

```
<taskdef name="pgcpackager" classname="com.ibm.ws.batch.packager.PGCPackager"
  classpath="${WAS_INSTALL_ROOT}/plugins/com.ibm.ws.batch.runtime.jar" />
```

6. After compiling the Java files in your application, invoke the WSBatchPackager task.

```
<WSBatchPackager appname="<appname>" earFile="<location name of EAR file to generate>"
  jarfile="location of the POJO jar file"/>
```

Results

You have developed a simple transactional batch application using a batch job controller and Enterprise JavaBeans (EJB) data stream, the command line, or the ANT tool.

What to do next

Install the compute-intensive application and configure WebSphere grid endpoints.

Components of a batch application

The batch application developer and the batch run time environment provide the components of a batch application.

The following tables describe the components of a batch application.

Table 40. Components of a batch application that are provided by the batch application developer. The table includes the component, type, implementation, and provider.

Component	Type	Implementation	Provider
Batch job step	POJO	com.ibm.websphere.BatchJobStepInterface	Batch application
Checkpoint algorithm	POJO	com.ibm.wsspi.batchCheckpointAlgorithm	Batch application (can use runtime-provided implementation instead)
Results algorithm	POJO	com.ibm.wsspi.batchResultsAlgorithm	Batch application (can use runtime-provided implementation instead)

Table 41. Components of a batch application that are provided by the batch run time environment. The table includes the component, type, implementation, and provider.

Component	Type	Implementation	Provider
Batch job controller	Session bean		Batch run time environment
Checkpoint algorithm	POJO		Batch run time environment (applications can provide their own)
Results algorithm	POJO		Batch run time environment (applications can provide their own)

Batch programming model

Batch applications are Enterprise JavaBeans (EJB) based Java Platform, Enterprise Edition (Java EE) applications. These applications conform to a few well-defined interfaces that allow the batch runtime environment to manage the start of batch jobs destined for the application.

Batch job steps

A batch job can be composed of one or more batch steps. All steps in a job are processed sequentially. Dividing a batch application into steps allows for separation of distinct tasks in a batch application. You can create batch steps by implementing the `com.ibm.websphere.batch.BatchJobStepInterface` interface. This interface provides the business logic of the batch step that the batch run time starts to run the batch application.

Batch controller bean

A batch application includes a stateless session bean that the product run time provides. This stateless session bean acts as a job step controller. The controller stateless session bean is declared in the application deployment descriptor once per batch application.

Batch data streams

Methods on the `BatchDataStream` interface allow the batch runtime environment to manage the data stream being used by a batch step. For example, one of the methods retrieves current cursor information from the stream to track how much data has been processed by the batch step.

Checkpoint algorithms

The batch runtime environment uses checkpoint algorithms to decide how often to commit global transactions under which batch steps are started. The XML Job Control Language (xJCL) definition of a batch job defines the checkpoint algorithms to be used. Properties specified for checkpoint algorithms in xJCL allow for checkpoint behavior, such as transaction timeouts and checkpoint intervals, to be customized for batch steps. The product provides time-based and record-based checkpoint algorithms. A checkpoint algorithm SPI is also provided for building additional custom checkpoint algorithms.

Results algorithm

Results algorithms are an optional feature of the batch programming model. Results algorithms are applied to batch steps through XML Job Control Language (xJCL). The algorithms are used to manipulate the return codes of batch jobs. Additionally, these algorithms are place holders for triggers based on step return codes.

Batch job return codes

Batch job return codes fall into two groups named *system* and *user*. System return codes are defined as negative integers. User application return codes are defined as positive integers. Both system and user ranges include the return code of zero (0). If a user application return code is specified in the system return code range, a warning message is posted in the job and system logs.

Batch job steps

You can separate tasks of a batch application into batch steps. Batch steps are implemented as Plain Old Java Object (POJO) classes that implement the interface, `com.ibm.websphere.batch.BatchJobStepInterface`. Batch job steps are performed sequentially.

Callback methods in the `BatchJobStepInterface` allow the grid endpoints to run batch steps when it runs a batch job.

A batch step contains the batchable business logic to run for a portion of the batch job. Typically, a batch step contains code to read a record from a batch data stream, perform business logic with that record and then continue to read the next record. The processJobStep method of a batch step class is called by the grid endpoints in a batch loop. This method contains all the logic that can be batched to perform on data.

The grid endpoints invoke batch step class methods in a global transaction. This global transaction is managed by the grid endpoints. The behavior of the transaction, such as transaction timeout or transaction commit interval, is controlled by the checkpoint algorithm associated with the batch job to which the step belongs.

The following grid endpoints callback methods exist on the BatchJobStepInterface that are invoked by the grid endpoints:

Table 42. Callback methods for grid endpoints. The table includes the callback method and a description.

Callback method	Description
setProperties(java.util.Properties properties)	Makes properties defined in XML Job Control Language (xJCL) available to batch step in a java.util.Properties object. This method is invoked in a global transaction.
void createJobStep()	Indicates to the step that it has been initialized. Initialization logic, such as retrieving a handle to a batch data stream, can be placed here. This method is invoked in a global transaction.
int processJobStep()	Repeatedly invoked by grid endpoints in a batch loop until the return code integer of this method indicates that the step has finished processing. Review BatchConstants in the batch API to see which return codes can be returned. A return code of BatchConstants.STEP_CONTINUE signals to the grid endpoints to continue calling this method in the batch loop. A return code of BatchConstants.STEP_COMPLETE indicates to the grid endpoints that the step has finished and to call destroyJobStep.
int destroyJobStep()	Indicates to the step that completion has occurred. The integer return code of this method is arbitrary and can be chosen by the batch application developer. This return code is saved in the grid endpoints database and represents the return code of the batch step. If the results algorithm is associated with the batch job, then this return code is passed to it. If there is a return code-based conditional logic in the xJCL of the batch job, then the grid endpoints use this return code to evaluate that logic.

The getProperties() method on the BatchJobStepInterface is not currently called by the grid endpoints. The method is included in the interface for symmetry and possible later use.

Batch return codes

The batch job return code is retrieved by using the getBatchJobRC EJB interface, the get BatchJobRC web services interface, or the lrcmd getBatchJobRC command option.

The following table lists the system batch job return codes that the batch environment uses. Do not confuse the batch job return code with either the job status constants (see the com.ibm.websphere.longrun.JobStatusConstants API) or the job scheduler constants (see the com.ibm.websphere.longrun.JobSchedulerConstants API). The JobStatusConstants represent the status of the job such as submitted, ended, restartable, canceled, or execution failed.

The job status can be obtained by using the getJobStatus EJB interface, the getJobStatus web services interface, or through the job management console. The JobSchedulerConstants represent operating conditions returned by the job scheduler on requests involving multiple jobs. For example:

```
int[] cancelJob( String[] jobid )
```

These conditions include:

1. Job does not exist
2. Job is in an invalid state
3. Database exception has occurred.

Table 43. Batch job return codes. The table includes each return code with an explanation.

Return code	Explanation
0	Job ended normally
-1	Internal protocol error - WSGrid utility

Table 43. Batch job return codes (continued). The table includes each return code with an explanation.

Return code	Explanation
-2	Input parameter error - WSGrid utility
-4	Job was suspended
-8	Job was canceled
-10	Job was forcibly canceled (z/OS only)
-12	Job failed and is in restartable state
-14	Job failed and is in execution failed state**
-16	Catastrophic failure - WSGrid utility

Note: This return code value does not apply in the case where the application returns `BatchConstants.STEP_COMPLETE_EXECUTION_FAILED` from the `processJobStep` method. In this case, the return code is determined by the application.

There are two options that are used to report an error in a batch application. The first option is for the application to produce an exception when an error is encountered. This results in termination of the job with a batch job return code of -12 and a batch job status of `restartable`. The second option is for the application to return a `BatchConstants.STEP_COMPLETE_EXECUTION_FAILED` return code (see the `com.ibm.websphere.batch.BatchConstants` API) from the `processJobStep` method and return an application-specific error return code from the `destroyJobStep` method. This results in termination of the job and a batch job status of `execution failed`. The step return code set in the `destroyJobStep` method is passed to any results algorithm specified on the job step and is used to influence the return code of the job to indicate the specific cause of the failure.

Batch controller bean

In its deployment descriptor, a batch application is required to declare a special stateless session bean. This bean acts as a batch job controller.

Each application can include only a single controller bean. You can only include a controller bean in a single work class, and a batch application can only have a single work class defined. This single work class is created when the application is installed. You can associate this work class with any service policy that has discretionary or queue-time goal type. The implementation of this bean is provided by the product, not by the batch application. The bean must be declared in the batch application deployment descriptor. Only one controller bean per batch application can be defined. The resource references and EJB references declared on the controller bean are available to batch data streams of the batch application in which the controller bean is declared. For example, if a batch data stream in the application needs access to a WebSphere Application Server data source, then a resource reference to that data source can be declared on the controller bean, and the batch data stream can look up the data source at run time in the `java:comp/env` name space.

Restrictions:

- The home interface must be `com.ibm.ws.batch.BatchJobControllerHome`.
- The remote interface must be `com.ibm.ws.batch.BatchJobController`.
- The EJB class must be `com.ibm.ws.batch.BatchJobControllerBean`.
- The transaction type can be `bean` or `container`.
- The session type must be `stateless`.
- There can be at most one batch controller stateless session bean per batch application.

The following example deployment descriptor illustrates a batch controller stateless session bean:

```
<session id="BatchController">
  <ejb-name>BatchController</ejb-name>
  <home>com.ibm.ws.batch.BatchJobControllerHome</home>
  <remote>com.ibm.ws.batch.BatchJobController</remote>
  <ejb-class>com.ibm.ws.batch.BatchJobControllerBean</ejb-class>
```

```

<session-type>Stateless</session-type>
<transaction-type>Bean</transaction-type>
<resource-ref id="ResourceRef_1117024737807">
  <description></description>
  <res-ref-name>wm/BatchWorkManager</res-ref-name>
  <res-type>commonj.work.WorkManager</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
</session>

```

Batch data streams

Batch data streams (BDS) are Java objects that provide an abstraction for the data stream processed by a batch step. A batch step can have one or more BDS objects associated with it. The grid endpoints make the BDS associated with the batch step available at run time. The grid endpoints also manages the life cycle of a BDS by invoking batch-specific callbacks.

A BDS object implements the `com.ibm.websphere.batch.BatchDataStream` interface. This interface is server agnostic. The implementing object can retrieve data from any type of data source, for example, files and databases. Call back methods on the `BatchDataStream` interface allow the grid endpoints to manage the BDS at run time. One of the key features of a BDS is its capability to convey its current position in the stream to the grid endpoints, and the capability to move itself to a given location in the data stream. This feature allows the grid endpoints to record (in the grid endpoints database) how much data a batch step has processed. This information is recorded on every checkpoint. Therefore, the grid endpoints can restart a batch job from a recorded position in the data stream if the job is canceled or fails in a recoverable manner.

The following lists the main methods that exist for the `BatchDataStream` interface. See the API for the `BatchDataStream` interface for additional information.

void open()

Called by grid endpoints to open the BDS

void close()

Called by grid endpoints to close the BDS

void initialize(String ilogicalname, String ijobstepid)

Called by grid endpoints to initialize the BDS and let it know its logical name and batch step ID

String externalizeCheckpointInformation()

Called by grid endpoints prior to a checkpoint to record the current cursor of the BDS

void internalizeCheckpointInformation(String chkpointInfo)

Called by grid endpoints to inform the BDS of the previously recorded cursor, `chkpointInfo`. Typically, the `positionAtCurrentCheckpoint` is called after this call to move the BDS to this cursor.

void positionAtCurrentCheckpoint()

Called by grid endpoints after calling `internalizeCheckpointInformation` to move the BDS to the cursor indicated by the `chkpointInfo` passed in through the `internalizeCheckpointInformation` call

The `BatchDataStream` interface does not have methods for retrieving or writing data. There are no `getNextRecord` and `putNextRecord` methods defined on the interface that a batch step calls to read or write to the BDS object. Methods for passing data between the batch step and the BDS object are the responsibility of the implementing BDS object. Review the batch Samples that this product supports to see examples of how to implement batch data streams.

Transaction environment

All methods of a BDS object are called in a global transaction. There is no guarantee that any consecutive method calls made to a BDS object happen in the same transaction because the transaction is owned by the grid endpoints, not the BDS object.

Checkpoint algorithms

The grid endpoints use checkpoint algorithms to determine when to commit global transactions under which batch steps are invoked. These algorithms are applied to a batch job through the XML Job Control Language (xJCL) definition. Properties specified for checkpoint algorithms in xJCL allow for checkpoint behavior, such as transaction timeouts and checkpoint intervals, to be customized for batch steps. The product provides both a time-based checkpoint algorithm and a record-based algorithm, and defines a service provider interface (SPI) for building additional custom checkpoint algorithms.

On each batch step iteration of the `processJobStep` method, the common batch container consults the checkpoint algorithm applied to that step if it commits the global transaction or not. Call back methods on the checkpoint algorithms allow the common batch container to inform the algorithm when a global transaction is committed or started. This behavior enables the algorithm to track the global transaction life cycle. On each iteration of the `processJobStep` method, the common batch container calls the `ShouldCheckpointBeExecuted` callback method on the algorithm to determine if the transaction is committed. The algorithm controls the checkpoint interval through this method.

Review the batch API for the checkpoint algorithm SPI, located in the Information Center reference section, for the checkpoint algorithm SPI that you can use to create custom checkpoint algorithms. The class name is `com.ibm.wsspi.batch.CheckpointPolicyAlgorithm`.

The product supports two checkpoint algorithms: the time-based algorithm and the record based algorithm. Both are explained in the following sections.

Time-based algorithm

The time-based checkpoint algorithm commits global transactions at a specified time interval. The following example declares a time-based algorithm in xJCL:

```
<checkpoint-algorithm name="timebased">
  <classname>com.ibm.wsspi.batch.checkpointalgorithms.timebased</classname>
  <props>
    <prop name="interval" value="15" />
    <prop name="TransactionTimeOut" value="30" />
  </props>
</checkpoint-algorithm>
```

The units of `interval` and `TransactionTimeOut` properties in the previous example are expressed in seconds.

Record-based algorithm

The record-based checkpoint algorithm commits global transactions at a specified number of iterations of the `processJobStep` method of batch step. Each call to the `processJobStep` method is treated as iterating through one record. The `processJobStep` method can retrieve multiple records from a batch data stream on each call. However, for this checkpoint algorithm one record is the equivalent one call to the `processJobStep` method.

The following example declares a record-based algorithm in xJCL:

```
<checkpoint-algorithm name="recordbased">
  <classname>com.ibm.wsspi.batch.checkpointalgorithms.recordbased</classname>
  <props>
    <prop name="recordcount" value="1000" />
    <prop name="TransactionTimeOut" value="60" />
  </props>
</checkpoint-algorithm>
```

The unit of the `TransactionTimeOut` property in the previous example is expressed in seconds.

If not specified in xJCL, the default transaction timeout is 60 seconds and the default record count is 10000.

Applying a checkpoint algorithm to a batch step

Checkpoint algorithms are applied to a batch job through xJCL. You can declare multiple checkpoint algorithms in xJCL, and you can apply a different algorithm to each batch step. You can apply no more than one checkpoint algorithm to a batch step.

The following example applies checkpoint algorithms in xJCL:

```
<job name="PostingsSampleEar">
  <checkpoint-algorithm name="timebased">
    <classname>com.ibm.wsspi.batch.checkpointalgorithms.timebased</classname>
    <props>
      <prop name="interval" value="15" />
      <prop name=" TransactionTimeout" value="30" />
    </props>
  </checkpoint-algorithm>
  <checkpoint-algorithm name="recordbased">
    <classname>com.ibm.wsspi.batch.checkpointalgorithms.recordbased</classname>
    <props>
      <prop name="recordcount" value="1000" />
      <prop name="TransactionTimeout" value="60" />
    </props>
  </checkpoint-algorithm>
  <job-step name="Step1">
    <checkpoint-algorithm-ref name="timebased" />
  </job-step>
  <job-step name="Step2">
    <checkpoint-algorithm-ref name="recordbased" />
  </job-step>
</job>
```

Results algorithms

Results algorithms are an optional feature of the batch programming model.

A results algorithm allows for two types of actions to occur at the end of a batch step:

- To influence the return code of the batch job based on the return code of the batch step that just ended. There are two types of return codes: The return code of an individual batch step and the return code of the batch job to which the step belongs.
- To provide a place holder for triggers or actions to take based on various step return codes.

Results algorithms are applied to a batch job through XML Job Control Language (xJCL). These algorithms are declared in xJCL and then applied to batch steps.

At the end of a batch step, the grid endpoints check the xJCL of the batch job to determine which results algorithm to invoke. For each results algorithm specified, the grid endpoints pass to the algorithm the return code of the batch step, which is the integer returned by the `destroyJobStep` method of the step, and the current return code of the batch job in the grid endpoints database. The results algorithm can then act based on the return codes passed in. The algorithm then passes a return code for the batch job back to the grid endpoints, which is persisted to the grid endpoints database as the current return code of the batch job. This return code can be the same as the return code that the grid endpoints passed to the results algorithm initially, or the return code can be different, depending on logic coded into the results algorithm. If a results algorithm is not specified on a batch step, the job return code is that of the results algorithm from the previous step. If no results algorithms are specified, the job return code is zero (0).

A results algorithm system programming interface (SPI) is also provided, which you can use to write your own algorithms and apply them to batch jobs.

The jobsum results algorithm

The jobsum results algorithm returns the highest return code of job steps to the grid endpoints. For example, there are three steps in the job (step1, step2, step3) where the following conditions exist:

- step1 returned 5
- step2 returned 8
- step3 returned 2. The jobsum algorithm ensures that 8 is passed to the grid endpoints as the final return code of the job.

Example of applying a jobsum and custom results algorithm to steps

```
<job name="PostingSampleEar">

  <results-algorithms>
    <results-algorithm name="jobsum">
      <classname>com.ibm.wsspi.resultsalgorithms.jobsum</classname>
    </results-algorithm>

    <results-algorithm name="custom_algorithm">
      <classname>my_custom_algorithm</classname>
    </results-algorithm>
  </results-algorithms>

  <job-step name="Step1">
    <results-ref name="jobsum">
  </job-step>

  <job-step name="Step2">
    <results-ref name="custom_algorithm">
  </job-step>

</job>
```

Skip-record processing

Use skip-record processing to skip read and write record errors in transactional batch jobs. Specify skip-record policies in the xJCL.

Skip-record processing

Each batch data stream has its own skip-record policy configuration. You enable skip-record processing by specifying a non-zero value for the `com.ibm.batch.bds.skip.count` batch data stream property in the xJCL.

You can refine skip record processing by using the `com.ibm.batch.bds.skip.include.exception.class.<n>` property to specify what record errors to skip and the `com.ibm.batch.bds.skip.exclude.exception.class.<n>` property to specify what record errors not to skip. The two properties are mutually exclusive.

The batch framework tracks skip record processing on a per step basis in the local job status database. This tracking is done only for batch data streams from the batch framework. At the end of step processing, a message is written to the job log. The message indicates the number of records that were skipped per batch data stream and the number of records per second per batch data stream. The number of records per second might not match the number of records that were processed by the batch data stream. If the actual number of records processed took less than a second, the value of the records per second is extrapolated from the amount of time it took to process the actual number of records.

The following list contains each of the skip-record properties followed by a description.

com.ibm.batch.bds.skip.count

Specifies the number of records that a batch data stream can skip due to an error in reading or writing a record. After the limit is reached, the batch data stream does not skip any more read or write errors.

When an input record is skipped, the batch data stream moves to the next record and fetches it. Control does not return to the caller until a record is read successfully, an error that does not involve skipping records occurs, or the skip limit is reached.

When an output record is skipped, the batch data stream returns to the caller normally.

If the batch data stream suffers a read or write error after the skip limit is reached, then the read or write exception is returned to the caller. The record is not skipped.

If you register a skip listener with the batch data stream, the skip listener receives control on every skipped record. For skipped reads, the `SkipListener.onSkippedRead(Throwable t)` method is invoked. The skip listener receives control before the next record is fetched. For skipped writes, the `SkipListener.onSkippedWrite(Object record, Throwable t)` method is invoked. The skipped record is passed in the first argument. The skip listener receives control before the batch data stream returns to the caller.

The running skip count for a batch data stream persists at every checkpoint. When a job step starts again, the skip count is restored from the last committed checkpoint.

Skip-record processing is disabled by default.

Any batch data stream implementation that extends the `com.ibm.websphere.batch.devframework.datastreams.bdsadapter.AbstractBatchDataInputStreamRecordMetrics` class or the `com.ibm.websphere.batch.devframework.datastreams.bdsadapter.AbstractBatchDataOutputStreamRecordMetrics` class automatically inherits skip-record support. All batch data streams defined under the `com.ibm.websphere.batch.devframework.datastreams` package contain skip-record support.

com.ibm.batch.bds.skip.include.exception.class.<n>

Specifies a list of exceptions for the batch data stream to skip when reading or writing records. The batch data stream skips only the exceptions on the list.

The `<n>` variable is an integer. Start the variable at 1 and increment it by one for each exception.

If you do not specify any exceptions, then the default is that all exceptions are included in the list of read/writer errors to skip.

The following example uses the property:

```
<batch-data-streams>
  <bds>
    <logical-name>inputBDS</logical-name>
    <props>
      <prop name="PATTERN_IMPL_CLASS" value="com.ibm.ws.batch.sample.bds.WCGSampleBDS"/>
      <prop name="file.encoding" value="8859_1"/>
      <prop name="FILENAME" value="/tmp/input.txt" />
      <prop name="com.ibm.batch.bds.skip.count" value="5" />
      <prop name="com.ibm.batch.bds.skip.include.exception.class.1"
        value="java.io.IOException" />
      <prop name="com.ibm.batch.bds.skip.include.exception.class.2"
        value="com.xyz.bds.error.BadDataException" />
    </props>
    <impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.TextFileReader</impl-class>
  </bds>
</batch-data-streams>
```

The batch data stream skips records for input/output exceptions and for bad data exceptions.

com.ibm.batch.bds.skip.exclude.exception.class.<n>

Specifies a list of exceptions that cannot be skipped when reading or writing records.

The `<n>` variable is an integer. Start the variable at 1 and increment it by one for each exception.

If you do not specify any exceptions, then the default is that no records are excluded from the list of read/write record errors to skip.

The following example uses the property:

```
<batch-data-streams>
  <bds>
    <logical-name>inputBDS</logical-name>
    <props>
      <prop name="PATTERN_IMPL_CLASS" value="com.ibm.ws.batch.sample.bds.WCGSampleBDS"/>
      <prop name="file.encoding" value="8859_1"/>
      <prop name="FILENAME" value="/tmp/input.txt" />
      <prop name="com.ibm.batch.bds.skip.count" value="3" />
      <prop name="com.ibm.batch.bds.skip.exclude.exception.class.1"
        value="java.io.FileNotFoundException" />
    </props>
    <impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.TextFileReader</impl-class>
  </bds>
</batch-data-streams>
```

The batch data stream does not skip records for file not found exceptions.

Skip listeners

You can register a skip listener with a batch data stream to listen for skipped records. The skip listener receives control whenever a record is skipped.

The following example code registers the skip listener.

```
AbstractBatchDataInputStream _inputBDS =
    (AbstractBatchDataInputStream)BatchDataStreamMgr.getBatchDataStream("inputBDS", getJobStepID());
((AbstractBatchDataStreamRecordMetrics)_inputBDS).addSkipListener(new MySkipListener());
```

Retry-step processing

Use retry-step processing to try job steps again when the processJobStep method encounters errors in a transactional batch job. Specify retry-step policies in the xJCL.

Each job step has its own retry-step policy configuration. You enable retry-step processing by specifying a non-zero value for the com.ibm.batch.step.retry.count job step property in the xJCL.

You can refine retry-step processing by using the com.ibm.batch.step.retry.include.exception.class.<n> property to specify what exceptions can be tried again when a step fails and the com.ibm.batch.step.retry.exclude.exception.class.<n> property to specify what exceptions cannot be tried again when a step fails. The two properties are mutually exclusive.

The batch framework tracks retry-step processing on a per step basis in the local job status database. At the end of step processing, a message is written to the job log. The message indicates the number of times that the step was tried again and the total clock time that the step used. The format of the clock time is HH:MM:SS:MMM where HH is hours, MM is minutes, SS is seconds, and MMM is milliseconds.

The following list contains the retry-step properties followed by a description.

com.ibm.batch.step.retry.count

Specifies the number of times a step can be tried again due to an error in step processing for the processJobStep method. When the limit is reached, no further step errors are tried again.

The BatchJobStepInterface.processJobStep method supports the throws java.lang.Exception clause. Any exception from the processJobStep method is eligible for retry-step processing.

Trying a step again is equivalent to restarting it. The BatchJobStepInterface.destroyJobStep method is called after the step error. The checkpoint transaction is rolled back before restarting the step. The BatchJobStepInterface.createJobStep method is called when a step is tried again. All batch data streams associated with the step are closed and reopened upon trying again.

If an error occurs for the step after the limit for trying a step again is reached, then the step fails and the job ends in the restartable state.

If you register a retry listener with the job step context, the retry listener receives control on every exception that can be tried again. The `RetryListener.onError(Throwable t)` method is called before the failed step enters the `destroyJobStep` method and before the checkpoint transaction is rolled back. The `RetryListener.onRetry(Throwable t)` method receives control when the step is tried again, but before the `BatchJobStepInterface.createJobStep` method is called.

The retry listener is unregistered immediately after the `RetryListener.onRetry` method is called. If you want the batch application to listen for further attempts to try the step again, reregister the retry listener.

The running count of the number of times a step can be tried again is reset to zero at every checkpoint. This means that the retry limit is effectively a per-checkpoint limit.

Retry-step processing is disabled by default.

com.ibm.batch.step.retry.delay.time

Specifies the number of milliseconds to wait before trying the step again. The delay occurs after the failed step goes through the `destroyJobStep` method and after the checkpoint transaction is rolled back. However, the delay occurs before calling the `RetryListener.onRetry` method.

com.ibm.batch.step.retry.include.exception.class.<n>

Specifies a list of exceptions that can be tried again when a step fails.

The `<n>` is an integer. Start the variable at 1 and increment it by one for each exception.

If you do not specify any exceptions, then the default is that all exceptions are included in the list.

The following example uses the property:

```
<job-step name="WCGStep1">
  <classname>com.ibm.ws.batch.sample.WCGSampleBDSBatchStep</classname>
  <checkpoint-algorithm-ref name="chkpt"/>
  <results-ref name="jobsum"/>
  <props>
    <prop name="com.ibm.batch.step.retry.count" value="1" />
    <prop name="com.ibm.batch.step.retry.delay.time" value="3000" />
    <prop name="com.ibm.batch.step.retry.include.exception.class.1" value="java.sql.SQLException" />
  </props>
  ...
</job-step>
```

The `WCGStep1` job step tries a job step again for a Structured Query Language (SQL) exception.

com.ibm.batch.step.retry.exclude.exception.class.<n>

Specifies a list of exceptions that cannot be tried again when a step fails.

The `<n>` variable is an integer. Start the variable at 1 and increment it by one for each exception.

If you do not specify any exceptions, then the default is that no exceptions are excluded from the list.

The following example uses the property:

```
<job-step name="WCGStep1">
  <classname>com.ibm.ws.batch.sample.WCGSampleBDSBatchStep</classname>
  <checkpoint-algorithm-ref name="chkpt"/>
  <results-ref name="jobsum"/>
  <props>
    <prop name="com.ibm.batch.step.retry.count" value="1" />
    <prop name="com.ibm.batch.step.retry.delay.time" value="3000" />
    <prop name="com.ibm.batch.step.retry.exclude.exception.class.1" value="java.sql.SQLException" />
  </props>
  ...
</job-step>
```

The WCGStep1 job step does not try a job step again for a Structured Query Language (SQL) exception.

Retry listeners

You can register a retry listener with the `JobStepContext` method to listen for exception to try again. The retry listener receives control whenever an exception that can be tried again occurs and the step is tried again.

The retry listener can be registered with the `JobStepContext` method through the `addRetryListener` method:

```
JobStepContextMgr.getContext().addRetryListener(new MyRetryListener());
```

Configurable transaction mode

Use the transaction mode to define whether job-related artifacts are called in global transaction mode or local transaction mode. Specify the transaction mode in the xJCL.

The following list describes the options for the `com.ibm.websphere.batch.transaction.policy` property.

global

Specifies that all job-related artifacts including listeners, batch data streams, and checkpoint algorithms, are called in global transaction mode. This option is the default.

local Specifies that all job-related artifacts including listeners, batch data streams, and checkpoint algorithms, are called in local transaction mode.

Updates to the local job status table and the database must be done through the same connection to maintain transaction integrity.

Transaction mode xJCL example

Specify a job level property:

```
<job .....  
...  
<props>  
  <prop name="com.ibm.websphere.batch.transaction.policy" value="LOCAL"/>  
</props>  
.....
```

Because the local transaction mode is specified, all user code must share the Java Database Connectivity (JDBC) connection with the batch container. The batch container creates and saves a JDBC connection in the job step context. You can fetch the job step context with the following code:

```
java.sql.Connection conn = JobStepContextMgr.getContext().getSharedSQLConnection();
```

Do not attempt to create a JDBC connection or close a connection obtained with the job step context. Local transaction support is built into the batch data stream framework JDBC reader or JDBC writer patterns. Setting the `com.ibm.websphere.batch.transaction.policy` property to `local` forces the batch data stream framework to use the shared JDBC connection.

Developing a parallel job management application

You can build a transactional batch application as a job and divide the job into subordinate jobs so that the subordinate jobs can run independently and in parallel. You use the parallel job manager to submit and manage the transactional batch jobs.

Before you begin

Note: Parallel job management applications built for prior versions of WebSphere Batch can run as is on WebSphere Application Server Version 8.5. However, follow the same procedure as having two xJCL files, an `xd.spi.properties` file, and a shared library configured for two SPI implementations. When using the application as is, do not do the steps in the following procedure.

Note: You can migrate parallel job management applications to WebSphere Application Server Version 8.5. Add the API implementation classes to the application EAR. Reauthor the xJCL as described in the following procedure.

About this task

You develop subordinate jobs for your job so that the subordinate jobs can run independently and in parallel. First develop the criteria that breaks jobs into subordinate jobs. Then build a simple batch application and make the pieces of the application parallel. Finally, create the xJCL for the subordinate jobs so that in another procedure you can deploy the application following another procedure.

Procedure

1. Develop the criteria that breaks jobs into subordinate jobs.

For example, divide jobs into subordinate jobs based on bank branches, with the subordinate job for each branch based on the location of the branch. For a given bank branch, have the subordinate job compute the balance of each account at the end of the day based on the daily transactions for each account.

2. Build a simple batch application to process the information as a job.
3. Make the pieces of the application parallel.

Implement the parameter API to divide the job for the batch application into multiple subordinate jobs.

4. Optional: Collect information about a subordinate job that is running.
Use the SubJobCollector API to collect the information.
5. Optional: Analyze information collected about the subordinate job.
Use the SubJobAnalyzer API to analyze the information.

6. Create the xJCL.

Start with xJCL from the job that you created for the simple batch application.

- a. Specify the run element as a child of the job element.

- 1) Set the instances attribute on the run element to `multiple`.
- 2) Set the Java virtual machine (JVM) attribute on the run element to `single` or `multiple`.

Use the `single` attribute to run all subordinate jobs in the same JVM. Use the `multiple` attribute to run the subordinate jobs on any valid JVM.

```
<run instances="multiple" jvm="multiple">
```

You can specify the run element as a child of a step. In this situation, the step is run as a parallel job. The contents of the step xJCL are used to generate a one step subordinate job xJCL.

- b. Specify one prop element as a child of the run element for each PJM API.

- 1) Specify the PJM API on the name attribute.

The following APIs are valid for the PJM:

- `com.ibm.websphere.batch.parallel.parameterizer`
- `com.ibm.websphere.batch.parallel.synchronization`
- `com.ibm.websphere.batch.parallel.subjobanalyzer`
- `com.ibm.websphere.batch.parallel.subjobcollector`

- 2) Set the value attribute to a name for the API.

- c. Include a prop element to specify the subordinate job name.

- 1) Set the name attribute to `com.ibm.wsspi.batch.parallel.subjob.name`.
 - 2) Set the value attribute to the name of the job.
- d. Optional: Include a `prop` element to indicate the job count.
- 1) Set the name attribute to `parallel.jobcount`.
 - 2) Set the value attribute to a value for the job count.
- e. Specify any other job-level properties.

```
<run instances="multiple" jvm="multiple">
  <props>
    <prop name="com.ibm.websphere.batch.parallel.parameterizer"
      value="com.ibm.websphere.samples.spi.MailerParameterizer"/>
    <prop name="com.ibm.websphere.batch.parallel.synchronization"
      value="com.ibm.websphere.samples.spi.MailerTXSynchronization"/>
    <prop name="com.ibm.websphere.batch.parallel.subjobanalyzer"
      value="com.ibm.websphere.samples.spi.MailerSubJobAnalyzer"/>
    <prop name="com.ibm.websphere.batch.parallel.subjobcollector"
      value="com.ibm.websphere.samples.spi.MailerSubJobCollector"/>

    <prop name="com.ibm.wsspi.batch.parallel.subjob.name"
      value="MailerSampleSubJob" />

    <!-- The count of parallel sub jobs to be submitted -->
    <prop name="parallel.jobcount" value="3" />

  </props>
</run>
```

7. Include the same three step level properties in each step in the xJCL.

```
<prop name="com.ibm.wsspi.batch.parallel.jobname" value="{parallel.jobname}" />
<prop name="com.ibm.wsspi.batch.parallel.logicalTXID" value="{logicalTXID}" />
<prop name="com.ibm.wsspi.batch.parallel.jobmanager" value="{parallel.jobmanager}" />
```

Results

You have created a job with subordinate jobs that can run independently and in parallel.

What to do next

Deploy the application as you would other batch applications.

Parallel job manager (PJM)

The parallel job manager (PJM) provides a facility and framework for submitting and managing transactional batch jobs that run as a coordinated collection of independent parallel subordinate jobs.

PJM basics

- The parallel job manager is in the batch container instead of in a separate system application.
- Only a single xJCL file is required. The xJCL combines the contents of the top-level job xJCL with the contents of the subordinate job xJCLs.
- You do not need to create a separate database.
- Because the PJM is part of the batch container, you do not need to install and configure the PJM.
- You package the PJM APIs in the batch application as a utility Java archive (JAR). No shared library is required.
- The contents of the `xd.spi.properties` file are part of the xJCL. No `xd.spi.properties` file is required.

The PJM operation and invocation of the APIs

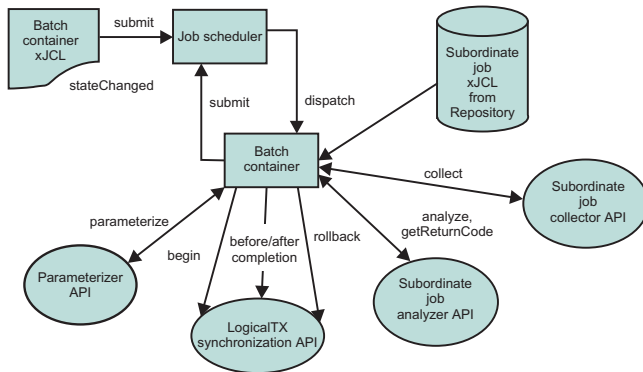
The following two images depict the PJM architecture and the sequence of a parallel job. First, the xJCL is submitted to the job scheduler. The job scheduler dispatches the xJCL to an endpoint that runs the application that the xJCL references. The batch container determines that the job is to have subordinate

jobs running in parallel from inspecting the run property of the job in the xJCL. The batch container delegates the running to the PJM subcomponent. The PJM invokes the parameterizer API and uses the information in the xJCL to help divide the job into subordinate jobs. The PJM then invokes the LogicalTX synchronization API to indicate the beginning of the logical transaction. The PJM generates the subordinate job xJCL and submits the subordinate jobs to the job scheduler. The job scheduler dispatches the subordinate jobs to the batch container endpoints so that they can run. The batch container runs the subordinate job. When a checkpoint is taken, the subordinate job collector API is invoked. This API collects relevant state information about the subordinate job. This data is sent to the subordinate job analyzer API for interpretation. After all subordinate jobs reach a final state, the beforeCompletion and afterCompletion synchronization APIs are invoked. The analyzer API is also invoked to calculate the return code of the job.

A logical transaction is a unit of work demarcation that spans the running of a parallel job. Its life cycle corresponds to the combined life cycle of the subordinate jobs of the parallel job. Because of an extension mechanism, you can customize application-managed resources so that they can be controlled in this unit of work scope for commit and rollback purposes.

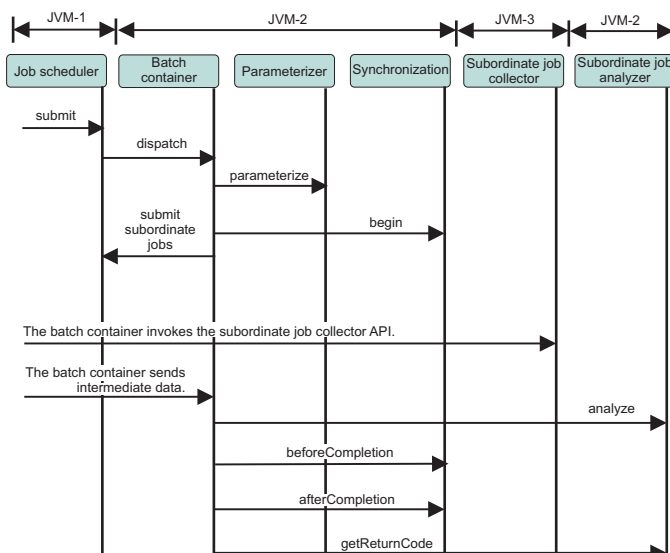
PJM architecture and programming model

The following image summarizes the PJM architecture, which shows where the APIs are called:



Sequence of a parallel job

The following image shows the order of events in a parallel job:



PJM job management

The top-level job submits the subordinate jobs and monitors their completion. The top-level job end state is influenced by the outcome of the subordinate jobs as follows:

1. If all subordinate jobs complete in the ended state, that is, in a successful completion, then the top-level job completes in the ended state.
2. If any subordinate job completes in the restartable state and no subordinate job has ended in the failed state, then the top-level job completes in the restartable state.
3. If any subordinate job completes in the failed state, then the top-level job completes in the failed state.
4. If the top-level job and subordinate jobs are in the restartable state, restart only the top-level job. If any subordinate jobs are restarted manually, then the top-level job does not process the logical transaction properly.

Parallel job manager application programming interfaces (APIs)

Parallel job manager (PJM) SPIs in previous releases are now APIs. They are packaged as part of an application.

Parameterizer API

The purpose of the parameterizer API is to divide the top-level job into multiple subordinate jobs. The parameterizer API determines the number of subordinate jobs to create, and the input properties passed to each subordinate job. Typically, the input properties contain information for which data chunks process a particular subordinate job. Implementation of the parameterizer API is mandatory.

There is a default parameterizer API that provides the basic functions of the generic parameterizer. To invoke this implementation, set the `com.ibm.websphere.batch.parallel.parameterizer` property to a value of `com.ibm.ws.batch.parallel.BuiltInParameterizer` in the xJCL. To specify the number of subordinate jobs, set the `com.ibm.wsspi.batch.parallel.jobs` input property to N, where N is the number of the subordinate job. To specify a unique property to a specific subordinate job instance, use the property `com.ibm.wsspi.batch.parallel.prop.<property_name>.<subordinate_job>=<value>` where *subordinate_job* is the subordinate job instance (1 < subordinate job < N). All other properties in `<property_name>=<value>` format are visible to all subordinate jobs.

Synchronization API

The synchronization API gives you control during the various life cycle stages of the logical transaction. For example, the `Begin`, `beforeCompletion`, and `afterCompletion` life stages. You can use these control points to roll back the logical transaction if necessary.

SubJobCollector API

The SubJobCollector API collects information related to a subordinate job execution. In a typical implementation, progress information about a subordinate job is stored as an externalizable object within the subordinate job context. When the batch container starts the collector API, the information previously stored within the subordinate job context is returned.

SubJobAnalyzer API

The SubJobAnalyzer API is used to analyze information collected previously by using the SubJobCollector API. In a typical implementation, the SubJobAnalyzer API is used to aggregate information obtained from all subordinate jobs to determine the consolidated return code for the top-level job. The SubJobAnalyzer API is called during checkpoint processing and on job completion.

Context objects

The batch runtime environment provides context objects that offer a common work area among APIs and batch application programming model artifacts. A context object allows user code to save and retrieve a Java Object and share it within the context scope. The batch runtime environment ensures appropriate cleanup of context objects and end of scope. There are two context object types:

- `ParallelJobManagerContext`: Exists in the scope of a parallel job. The `Parameters`, `SubJobAnalyzer`, and `Synchronization` APIs all have access to this context for a given parallel job instance.
- `SubJobContext`: Exists in the scope of a subordinate job. The `SubJobCollector`, and batch application programming model artifacts, `BatchDataStream`, `BatchJobStepInterface`, `CheckpointPolicyAlgorithm`, and `ResultsAlgorithm` all have access to this context for a given subordinate job instance.

Other considerations for the parallel job manager

There are other considerations that help you understand how to optimally use the parallel job manager.

Transaction timeouts

You can keep the default value for the `TransactionTimeout` property in the top-level job `xJCL`. You can alternatively adjust it depending on the transaction timeout of the subordinate job.

Job logs

You can view job logs for a subordinate job from the job management console. The PJM retrieves subordinate job logs for its logical job and aggregates them into its top-level job log.

Failover scenarios

If the top-level job and several subordinate jobs are in the restartable state, then restart only the top-level job. If you restart any of the subordinate jobs manually, then the top-level job does not process the logical transaction properly.

Disaster recovery

You can use the `jobrecovery` script when a primary site fails to allow a secondary site to take over.

Using the batch data stream framework

This topic shows you an example of how to use the batch data stream (BDS) framework.

Before you begin

Identify the correct pattern to use. Select a pattern based on what type of data stream you need. For example, if you want to read text from a file, then select the `FileReaderPattern`. See “Batch data stream framework and patterns” on page 138 for a selection of patterns.

Procedure

1. Implement the pattern interface:

```
<codeblock>package com.ibm.websphere.samples;

import java.io.BufferedReader;
import java.io.IOException;
import java.util.Properties;

import com.ibm.websphere.batch.devframework.configuration.BDSFWLogger;
import com.ibm.websphere.batch.devframework.datastreams.patternadapter.FileReaderPattern;

// Implement the FileReaderPattern
```

```

public class TransactionListStream implements FileReaderPattern {
    private Properties properties;
    private BDSFWLogger logger;

    /**
     * Save properties specified in the xJCL
     */

    public void initialize(Properties props) {
        // create logger
        logger = new BDSFWLogger(props);

        if (logger.isDebugEnabled())
            logger.debug("entering TransactionListInputStream.initialize()");
        properties = props;
    }

    // This method is where you should add the business logic of processing the read //string
    public Object fetchRecord(BufferedReader reader) throws IOException {
        String str = null;
        Posting posting = null;
        if (logger.isDebugEnabled())
            logger.debug("Entering TransactionListInputStream.fetchRecord");
        if(reader.ready()) {
            str = reader.readLine();
        }
        if(str != null) {

            posting = _generateRecord(str);

        }

        if (logger.isDebugEnabled())
            logger.debug("Exiting TransactionListInputStream.fetchRecord with " + posting);
        return posting;
    }

    // Helper method that parses the read string and creates an internal object for use
    // by other parts of the code
    private Posting _generateRecord(String str) {
        Posting post = null;
        String [] tokens = str.split(",", 3);

        if(tokens.length == 3) {

            String txTypeStr = tokens[0];
            String actNoStr = tokens[1];
            String amtStr = tokens[2];

            int txType = Integer.parseInt(txTypeStr);
            double amt = Double.parseDouble(amtStr);
            post = new Posting(txType,actNoStr,amt);

        } else {
            logger.error("Invalid csv string" + str);
        }
        if(logger.isDebugEnabled())
            logger.debug("Loaded posting record " + post);
        return post;
    }

    public void processHeader(BufferedReader reader) throws IOException {
        // NO OP for this sample
    }

}
</codeblock>

```

2. Reference the class that you created in the previous step, along with the supporting class in the xJCL.

xJCL example

```
<codeblock><batch-data-streams>
  <bds>
    <logical-name>txlististream</logical-name>
    <props>
      <prop name="IMPLCLASS" value="com.ibm.websphere.samples.TransactionListStream"/>
      <prop name="FILENAME" value="/opt/inputfile.txt"/>
      <prop name="debug" value="true"/>
    </props>
  <impl-class> com.ibm.websphere.batch.devframework.datastreams.patterns.TextFileReader </impl-class>
</bds>
</batch-data-streams>
</codeblock>
```

What to do next

Install the application.



Batch data stream framework and patterns

The batch environment provides a batch data stream (BDS) framework that includes pre-built code to work with streams such as text, byte, database, and data sets. You can implement an interface where the business logic for processing the stream is added. The pre-built code manages actions such as opening, closing, and externalizing and internalizing checkpoints.

BDS framework patterns

A BDS framework pattern is a simple Java™ interface for a particular type of data stream that a user implements to insert business logic. The BDS framework has several supporting classes for each pattern that do most of the mundane tasks related to stream management. The following table shows the patterns that the batch environment provides:

Table 44. BDS framework patterns. The table includes the pattern name, description, and supporting classes.

Pattern name	Description	Supporting classes
"JDBCReaderPattern" on page 139	Used to retrieve data from a database using a JDBC connection.	<ul style="list-style-type: none">LocalJDBCReaderJDBCReaderCursorHoldableJDBCReader
"JDBCWriterPattern" on page 141	Used to write data to a database using a JDBC connection.	<ul style="list-style-type: none">LocalJDBCWriterJDBCWriter
"ByteReaderPattern" on page 143	Used to read byte data from a file.	FileByteReader
"ByteWriterPattern" on page 144	Used to write byte data from a file.	FileByteWriter
"FileReaderPattern" on page 145	Used to read a text file.	TextFileReader
"FileWriterPattern" on page 147	Used to write to a text file.	TextFileWriter
 "RecordOrientedDatasetReaderPattern" on page 148	Used to read a z/OS data set.	<ul style="list-style-type: none">ZFileStreamOrientedTextReaderZFileStreamOrientedByteReaderZFileRecordOrientedDataReader
 "RecordOrientedDataSetWriterPattern" on page 150	Used to write to a z/OS data set.	<ul style="list-style-type: none">ZFileStreamOrientedTextWriterZFileStreamOrientedByteWriterZFileRecordOrientedDataReader
"JPAREaderPattern" on page 151	Used to retrieve data from a database using OpenJPA	JPAReader
"JPASWriterPattern" on page 153	Used to write data to a database using a Java Persistence API (JPA) connection.	JPAWriter

BDS framework steps

BDS framework steps minimize the amount of work to create a batch step by performing the bookkeeping tasks related to step management and delegating the business logic to a class implemented by the user.

Table 45. BDS framework steps. The table includes the framework step and description.

Step	Description
"Implementing the generic batch step (GenericXDBatchStep)" on page 157	A simple step that uses one input and one output stream.
"Implementing the error tolerant step" on page 158	A simple step that uses one input, one output stream, and one error stream.

ThresholdPolicies

Table 46. ThresholdPolicies. The table includes the step and description.

Step	Description
"Declaring the record based threshold policy (RecordBasedThresholdPolicy)" on page 160	This policy provides a batch implementation of the ThresholdPolicy interface.
"Declaring the percentage-based threshold policy (PercentageBasedThresholdPolicy)" on page 159	This policy provides a batch implementation of the ThresholdPolicy interface

JDBCReaderPattern

This pattern is used to retrieve data from a database using a Java Database Connectivity (JDBC) connection.

Supporting classes

1. CursorHoldableJDBCReader

This class is referenced when the usage pattern of your JDBC input stream retrieves a set of results at the beginning of the step, and then iterates over them throughout the step-processing logic. The CursorHoldableJDBCReader uses a stateful session bean with a cursor holdable, non-XA data source. A cursor holdable JDBCReader is a pattern that is implemented in such a way that the cursor is not lost when the transaction is committed. As a result, ResultSets do not need to be repopulated after every checkpoint, which improves performance. To use CursorHoldableJDBCReader, package the CursorHoldableSessionBean in your application. To create the package, add the `nonxadsjndiname=jndi_name_of_a_non-XA_data_source_to_database` property to the properties file that is used by the BatchPackager. For example, `nonxadsjndiname=jdbc/nonxads`. If you want to add multiple non-XA data sources enter the following: `nonxadsjndiname=<jndi name1>;<jndi name2>...`

Restriction: Currently, the resource reference name of the JDBC data source is the same as the Java Naming and Directory Interface (JNDI) name.

2. JDBCReader

This class is referenced when the usage pattern of your JDBC input stream retrieves a single result from a query, which is used and discarded after every iteration of the step.

3. LocalJDBCReader

This class is referenced when data is read from a local database.

Required properties

The following properties are required for the pattern.

Table 47. Required properties. The table includes each required property, its value, and whether the LocalJDBCReader class, the CursorHoldableJDBCReader class, or the JDBCReader class is applicable.

Property	Value	LocalJDBCReader	CursorHoldableJDBCReader	JDBCReader
PATTERN_IMPL_CLASS	Class implementing JDBCReaderPattern interface	Applicable	Applicable	Applicable
ds_jndi_name	Datasource JNDI name.	Applicable	Not applicable	Applicable

Table 47. Required properties (continued). The table includes each required property, its value, and whether the LocalJDBCReader class, the CursorHoldableJDBCReader class, or the JDBCReader class is applicable.

Property	Value	LocalJDBCReader	CursorHoldableJDBCReader	JDBCReader
jdbc_url	The JDBC URL. For example, jdbc:derby:C:\mysample\CREDITREPORT.	Applicable	Not applicable	Not applicable
jdbc_driver	The JDBC driver. For example, org.apache.derby.jdbc.EmbeddedDriver	Applicable	Not applicable	Not applicable
userid	The user ID for the database. For example, Myid	Applicable	Not applicable	Not applicable
pswd	User password. For example, mypwd. LocalJDBCReader only.	Applicable	Not applicable	Not applicable

Optional properties

The following properties are optional for the pattern.

Table 48. Optional properties. The table includes each optional property, its value and description, and whether the LocalJDBCReader class, the CursorHoldableJDBCReader class, or the JDBCReader class is applicable.

Property name	Value	Description	LocalJDBCReader	CursorHoldableJDCReader	JDBCReader
debug	true or false (default is false)	Enables detailed tracing on this batch datastream.	Applicable	Applicable	Applicable
EnablePerformanceMeasurement	true or false (default is false)	Calculates the total time spent in the batch data-streams and the processRecord method, if you are using the GenericXDBatchStep.	Applicable	Applicable	Applicable
EnableDetailedPerformanceMeasurement	true or false (default is false)	Provides a more detailed breakdown of time spent in each method of the batch data-streams.	Applicable	Applicable	Applicable

Interface definition

```
public interface JDBCReaderPattern {

    /**
     * This method is invoked during the job setup phase.
     *
     * @param props properties provided in the xJCL
     */
    public void initialize(Properties props);

    /**
     * This method should retrieve values for the various columns for the current row from the given resultset
     * object. Typically this data would be used to populate an intermediate object which would be returned
     * @param resultSet
     * @return
     */
    public Object fetchRecord(ResultSet resultSet);

    /**
     * This method should return a SQL query that will be used during setup of the stream to retrieve all
     * relevant data that would be processed part of the job steps
     * @return object to be used during process step.
     */
    public String getInitialLookupQuery();

    /**

```

```

* This method gets called during Job Restart. The restart token should be used to create an SQL query
* that will retrieve previously unprocessed records. Typically the restart token would be the primary
* key in the table and the query would get all rows with
* primary key value > restarttoken
* @param restartToken
* @return The restart query
*/
public String getRestartQuery(String restartToken);

/**
* This method gets called just before a checkpoint is taken.
* @return The method should return a string value identifying the last record read by the stream.
*/
public String getRestartTokens();
}

```

CursorHoldableJDBCReader xJCL example

```

<batch-data-streams>
<bds>
<logical-name>inputStream</logical-name>
<props>
<prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.batch.samples.tests.bds.EchoReader"/>
<prop name="ds_jndi_name" value="jdbc/fvtdb"/>
<prop name="debug" value="true"/>
<prop name="DEFAULT_APPLICATION_NAME" value="XDCGIVT"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.CursorHoldableJDBCReader</impl-class>
</bds>
</batch-data-streams>

```

LocalJDBCReader xJCL example

```

<batch-data-streams>
<bds>
<logical-name>inputStream</logical-name>
<props>
<prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.batch.samples.tests.bds.EchoReader"/>
<prop name="jdbc_url" value="jdbc:derby:C:\mysample\CREDITREPORT"/>
<prop name="jdbc_driver" value="org.apache.derby.jdbc.EmbeddedDriver"/>
<prop name="user_id" value="myuserid"/>
<prop name="pswd" value="myspwd"/>
<prop name="debug" value="true"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.LocalJDBCReader</impl-class>
</bds>
</batch-data-streams>

```

JDBCWriterPattern

The JDBCWriterPattern pattern is used to write data to a database using a JDBC connection.

Supporting classes

- JDBCWriter
- LocalJDBCWriter

Required properties

The following properties are required for the pattern.

Table 49. Required properties. The table includes the name and value of each required property for the pattern.

Property name	Value	LocalJDBCWriter	JDBCWriter
PATTERN_IMPL_CLASS	Class implementing JDBCWriterPattern interface	Applicable	Applicable
ds_jndi_name	Datasource JNDI name.	Applicable	Not applicable
jdbc_url	The JDBC URL. For example, jdbc:derby:C:\mysample\CREDITREPORT.	Applicable	Not applicable

Table 49. Required properties (continued). The table includes the name and value of each required property for the pattern.

Property name	Value	LocalJDBCWriter	JDBCWriter
jdbc_driver	The JDBC driver. For example, org.apache.derby.jdbc.EmbeddedDriver	Applicable	Not applicable
user_id	The user ID for the database. For example, Myid	Applicable	Not applicable
pswd	User password. For example, mypwd. LocalJDBCReader only.	Applicable	Not applicable

Optional properties

The following properties are optional for the pattern.

Table 50. Optional properties. The table includes the name, value, and description of each optional property for the pattern.

Property name	Value	Description	LocalJDBCReader	JDBCWriter
debug	true or false (default is false)	Enables detailed tracing on this batch datastream.	Applicable	Applicable
EnablePerformanceMeasurement	true or false (default is false)	Calculates the total time spent in the batch data-streams and the processRecord method, if you are using the GenericXDBatchStep.	Applicable	Applicable
EnableDetailedPerformanceMeasurement	true or false (default is false)	Provides a more detailed breakdown of time spent in each method of the batch data-streams.	Applicable	Applicable
batch_interval	Default value is 20. Make the value less than the checkpoint interval for record-based checkpointing.	Denotes the number of SQL updates to batch before committing.	Applicable	Applicable

Interface definition

```
public interface JDBCWriterPattern {

    public void initialize(Properties props);

    /**
     * This is typically an Update query used to write data into the DB
     * @return
     */
    public String getSQLQuery();

    /**
     * The parent class BDSJDBCWriter creates a new preparedstatement and
     * passes it to this method. This method populates the preparedstatement
     * with appropriate values and returns it to the parent class for execution
     * @param pstmt
     * @param record
     * @return
     */
    public PreparedStatement writeRecord(PreparedStatement pstmt, Object record);
}
```

JDBCWriter xJCL example

```
<batch-data-streams>
<bds>
<logical-name>outputStream</logical-name>
<props>
<prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.batch.samples.tests.bds.EchoWriter"/>
<prop name="ds_jndi_name" value="jdbc/fvtldb"/>
<prop name="debug" value="true"/>
```



```

</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.JDBCWriter</impl-class>
</bds>
</batch-data-streams>

```

LocalJDBCWriter xJCL example

```

<batch-data-streams>
<bds>
<logical-name>outputStream</logical-name>
<props>
<prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.batch.samples.tests.bds.EchoWriter"/>
<prop name="jdbc_url" value="jdbc:derby:C:\mysample\CREDITREPORT"/>
<prop name="jdbc_driver" value="org.apache.derby.jdbc.EmbeddedDriver"/>
<prop name="user_id" value="myuserid"/>
<prop name="pswd" value="mypasswd"/>
<prop name="debug" value="true"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.LocalJDBCWriter</impl-class>
</bds>
</batch-data-streams>

```

ByteReaderPattern

The ByteReaderPattern pattern is used to read byte data from a file.

Supporting classes

The FileByteReader class provides the logic for opening and reading byte data from the given file.

Required properties

The following properties are required for the pattern.

Table 51. Required properties. The table includes the name and value of each required property for the pattern.

Property name	Value
PATTERN_IMPL_CLASS	Class implementing ByteReaderPattern interface
FILENAME	Complete path to the input file

Optional properties

The following properties are optional for the pattern.

Table 52. Optional properties. The table includes the name, value, and description of each optional property for the pattern.

Property name	Value	Description
debug	true or false (default is false)	Enables detailed tracing on this batch data stream.
EnablePerformanceMeasurement	true or false (default is false)	Calculates the total time spent in the batch data streams and the processRecord method, if you are using the GenericXDBatchStep.
EnableDetailedPerformanceMeasurement	true or false (default is false)	Provides a more detailed breakdown of time spent in each method of the batch data streams.
file.encoding	Encoding of the file.	For example, 8859_1
AppendJobIdToFileName	true or false (default is false)	Appends the JobID to the file name before loading the file.

Interface definition

```

public interface ByteReaderPattern {

    /**
     * Is called by the framework during Step setup stage
     * @param props
     */
    public void initialize(Properties props);
}

```

```

/**
 *
 * @param reader
 * @throws IOException
 */

public void processHeader(BufferedInputStream reader) throws IOException;

/**
 * Get the next record from the input stream
 * @param reader
 * @return
 * @throws IOException
 */
public Object fetchRecord(BufferedInputStream reader) throws IOException;
}

```

xJCL example

```

<batch-data-streams>
<bds>
<logical-name>inputStream</logical-name>
<props>
<prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.batch.samples.tests.bds.EchoReader"/>
<prop name="file.encoding" value="8859_1"/>
<prop name="FILENAME" value="/opt/txlist.txt" />
<prop name="debug" value="true"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteReader</impl-class>
</bds>
</batch-data-streams>

```

ByteWriterPattern

The ByteWriterPattern pattern is used to write byte data to a file.

Supporting classes

The FileByteWriter class provides the logic to open and write bytes to the given file. It can either append or overwrite existing content, depending on the properties specified. During a restart, the file is always opened in append mode.

Required properties

The following properties are required for the pattern.

Table 53. Required properties. The table includes the name and value of each required property for the pattern.

Property name	Value
PATTERN_IMPL_CLASS	Class implementing ByteWriterPattern interface
FILENAME	Complete path to the input file

Optional properties

The following properties are optional for the pattern.

Table 54. Optional properties. The table includes the name, value, and description of each optional property for the pattern.

Property name	Value	Description
debug	true or false (default is false)	Enables detailed tracing on this batch datastream.
EnablePerformanceMeasurement	true or false (default is false)	Calculates the total time spent in the batch data-streams and the processRecord method, if you are using the GenericXDBatchStep.

Table 54. Optional properties (continued). The table includes the name, value, and description of each optional property for the pattern.

Property name	Value	Description
EnableDetailedPerformanceMeasurement	true or false (default is false)	Provides a more detailed breakdown of time spent in each method of the batch data-streams.
file.encoding	Encoding of the file	For example, 8859_1
AppendJobIdToFileName	true or false (default is false)	Appends the JobID to the file name before loading the file.
append	true or false (default is true)	Determines whether to open the file in append mode. Important: During a restart, the file is always opened in append mode.

Interface definition

```
public interface ByteWriterPattern {

    /**
     * Invoked during the step setup phase
     * @param props
     */
    public void initialize(Properties props);

    /**
     * Writes the given object onto the given outputstream. Any processing
     * that needs to be done before writing can be added here
     * @param out
     * @param record
     * @throws IOException
     */
    public void writeRecord(BufferedOutputStream out, Object record) throws IOException;

    /**
     * Write header information if any
     * @param out
     * @throws IOException
     */
    public void writeHeader(BufferedOutputStream out) throws IOException;

    /**
     * This method can be optionally called during process step to explicitly
     * initialize and write the header.
     * @param header
     */
    public void writeHeader(BufferedOutputStream out, Object header) throws IOException;
}
```

xJCL example

```
<batch-data-streams>
<bds>
<logical-name>outputStream</logical-name>
<props>
  <prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.batch.samples.tests.bds.EchoWriter"/>
  <prop name="file.encoding" value="8859_1"/>
  <prop name="FILENAME" value="/opt/txlist.txt" />
  <prop name="debug" value="true"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteWriter</impl-class>
</bds>
</batch-data-streams>
```

FileReaderPattern

The FileReaderPattern pattern is used to read text data from a file.

Supporting classes

The `TextFileReader` class provides the logic to open and read text data line by line.

Required properties

The following properties are required for the pattern.

Table 55. Required properties. The table includes the name and value of each required property for the pattern.

Property name	Value
<code>PATTERN_IMPL_CLASS</code>	Class implementing <code>FileReaderPattern</code> interface
<code>FILENAME</code>	Complete path to the input file

Optional properties

The following properties are optional for the pattern.

Table 56. Optional properties. The table includes the name, value, and description of each optional property for the pattern.

Property name	Value	Description
<code>debug</code>	true or false (default is false)	Enables detailed tracing on this batch datastream.
<code>EnablePerformanceMeasurement</code>	true or false (default is false)	Calculates the total time spent in the batch data-streams and the <code>processRecord</code> method, if you are using the <code>GenericXDBatchStep</code> .
<code>EnableDetailedPerformanceMeasurement</code>	true or false (default is false)	Provides a more detailed breakdown of time spent in each method of the batch data-streams.
<code>file.encoding</code>	Encoding of the file.	For example, <code>8859_1</code>
<code>AppendJobIdToFileName</code>	true or false (default is false)	Appends the JobID to the file name before loading the file.

Interface definition

```
public interface FileReaderPattern {

    /**
     * Invoked during the step setup phase
     * @param props
     */
    public void initialize(Properties props);

    /**
     * This method is invoked only once. It should be used
     * to read any header data if necessary.
     * @param reader
     * @throws IOException
     */
    public void processHeader(BufferedReader reader) throws IOException;

    /**
     * This method should read the next line from the reader
     * and return the data in suitable form to be processed
     * by the step.
     * @param reader
     * @return
     * @throws IOException
     */
    public Object fetchRecord(BufferedReader reader) throws IOException;

    /**
     * This method can be optionally invoked from the process step
     * to obtain the header data that was previously obtained during the processHeader
     * call
     * @return
     */
}
```

```

*/
    public Object fetchHeader();
}

```

xJCL example

```

<batch-data-streams>
<bds>
<logical-name>inputStream</logical-name>
<props>
  <prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.batch.samples.tests.bds.EchoReader"/>
  <prop name="file.encoding" value="8859_1"/>
  <prop name="FILENAME" value="/opt/txlist.txt" />
  <prop name="debug" value="true"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.TextFileReader</impl-class>
</bds>
</batch-data-streams>

```

FileWriterPattern

The FileWriterPattern pattern is used to write text data to a file.

Supporting classes

The TextFileWriter class provides the logic to open and write string data to the given file. The file is opened either in append or overwrite mode, depending on the properties specified. The file is always opened in append mode during a job restart.

Required properties

The following properties are required for the pattern.

Table 57. Required properties. The table includes the name and value of each required property for the pattern.

Property name	Value
PATTERN_IMPL_CLASS	Class that implements the FileWriterPattern interface
FILENAME	Complete path to the input file

Optional properties

The following properties are optional for the pattern.

Table 58. Optional properties. The table includes the name, value, and description of each optional property for the pattern.

Property name	Value	Description
debug	true or false (default is false)	Enables detailed tracing on this batch datastream.
EnablePerformanceMeasurement	true or false (default is false)	Calculates the total time spent in the batch data-streams and the processRecord method, if you are using the GenericXDBatchStep.
EnableDetailedPerformanceMeasurement	true or false (default is false)	Provides a more detailed breakdown of time spent in each method of the batch data-streams.
file.encoding	Encoding of the file	For example, 8859_1
AppendJobIdToFileName	true or false (default is false)	Appends the JobID to the file name before loading the file.
append	true or false (default is true)	Determines whether to open the file in append mode. Important: During a restart, the file is always opened in append mode.

Interface definition

```
public interface FileWriterPattern {

    /**
     * Invoked during step setup phase
     * @param props
     */
    public void initialize(Properties props);

    /**
     * This method should write the given record
     * object to the bufferedwriter.
     * @param out
     * @param record
     * @throws IOException
     */
    public void writeRecord(BufferedWriter out, Object record) throws IOException;

    /**
     * This method is invoked only once just after the bufferedwriter
     * is opened. It should be used to write any header information
     * @param out
     * @throws IOException
     */
    public void writeHeader(BufferedWriter out) throws IOException;

    /**
     * This method can be optionally called during process step to explicitly
     * initialize and write the header.
     * @param header
     * @throws IOException
     */
    public void writeHeader(BufferedWriter out, Object header) throws IOException;

}
```

xJCL sample

```
<batch-data-streams>
<bds>
<logical-name>outputStream</logical-name>
<props>
  <prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.batch.samples.tests.bds.EchoWriter"/>
  <prop name="file.encoding" value="8859_1"/>
  <prop name="FILENAME" value="/opt/txlist.txt" />
  <prop name="debug" value="true"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.TextFileWriter</impl-class>
</bds>
</batch-data-streams>
```

RecordOrientedDatasetReaderPattern

The RecordOrientedDatasetReaderPattern pattern is used to read data from a z/OS data set.

Supporting classes

- ZFileStreamOrientedTextReader: Reads text data
- ZFileStreamOrientedByteReader: Reads byte data
- ZFileRecordOrientedDataReader: Reads sequential data

Required properties

The following properties are required for the pattern.

Table 59. Required properties. The table includes the name and value of each required property for the pattern.

Property name	Value	Description
PATTERN_IMPL_CLASS	Java class name	Class that implements the RecordOrientedDatasetReaderPattern interface

Table 59. Required properties (continued). The table includes the name and value of each required property for the pattern.

Property name	Value	Description
DSNAME	Dataset name	For example, <i>USER216.BATCH.RECORD.OUTPUT</i>

Optional properties

The following properties are optional for the pattern.

Table 60. Optional properties. The table includes the name, value, and description of each optional property for the pattern.

Property name	Value	Description
ds_parameters	Parameters used to open the data set.	Default for ZFileRecordOrientedDataReader is <i>rb,recfm=fb,type=record,lrecl=80</i> and Default for ZFileStreamOrientedByteReader and ZFileStreamOrientedTextReader is <i>rt</i>
debug	true or false (default is false)	Enables detailed tracing on this batch datastream.
EnablePerformanceMeasurement	true or false (default is false)	Calculates the total time spent in the batch data-streams and the processRecord method, if you are using the GenericXDBatchStep.
EnableDetailedPerformanceMeasurement	true or false (default is false)	Provides a more detailed breakdown of time spent in each method of the batch data-streams.
file.encoding	Encoding of the file.	For example, <i>8859_1</i> .

Interface definition

```
public interface RecordOrientedDatasetReaderPattern {

    /**
     * This method is invoked during the job setup phase.
     * The properties are the ones specified in the xJCL.
     * @param props
     */
    public void initialize(Properties props);

    /**
     * This method is invoked only once immediately after
     * the Zfile is opened. It should be used to process
     * header information if any.
     * @param reader
     * @throws IOException
     */
    public void processHeader(ZFile reader) throws IOException;

    /**
     * This method should read the next record from the Zfile
     * and return it in an appropriate form (as an intermediate object)
     * @param reader
     * @return
     * @throws IOException
     */
    public Object fetchRecord(ZFile reader) throws IOException;
}
```

xJCL example

```
<batch-data-streams>
<bds>
<logical-name>inputStream</logical-name>
<props>
  <prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.batch.samples.tests.bds.EchoReader"/>
  <prop name="DSNAME" value="USER216.BATCH.RECORD.INPUT"/>
  <prop name="ds_parameters" value="rt"/>
  <prop name="file.encoding" value="CP1047"/>
  <prop name="debug" value="true"/>
</props>
```

```

<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.ZFileStreamOrientedByteReader</impl-class>
</bds>
  <bds>
    <logical-name>outputStream</logical-name>
    <props>
      <prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.batch.samples.tests.bds.EchoWriter"/>
      <prop name="DSNAME" value="USER216.BATCH.RECORD.OUTPUT"/>
      <prop name="ds_parameters" value="wt"/>
      <prop name="file.encoding" value="CP1047"/>
      <prop name="debug" value="\${debug}"/>
    </props>
  </bds>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.ZFileStreamOrientedByteWriter</impl-class>
</bds>
</batch-data-streams>

```

RecordOrientedDataSetWriterPattern

The RecordOrientedDataSetWriterPattern pattern is used to write data to a z/OS data set.

Supporting classes

- ZFileStreamOrientedTextWriter: Writes text data
- ZFileStreamOrientedByteWriter: Writes byte data
- ZFileRecordOrientedDataWriter: Writes sequential data

Required properties

The following properties are required for the pattern.

Table 61. Required properties. The table includes the name, value, and description of each required property for the pattern.

Property name	Value	Description
PATTERN_IMPL_CLASS	Java class name	Class implementing RecordOrientedDatasetWriterPattern interface
DSNAME	Data set name	For example, USER216.BATCH.RECORD.OUTPUT

Optional properties

The following properties are optional for the pattern.

Table 62. Optional properties. The table includes the name, value, and description of each optional property for the pattern.

Property name	Value	Description
ds_parameters	Parameters used to open the data set.	Default for ZFileRecordOrientedDataWriter is wb,recfm=fb,type=record,lrec1=80 and Default for ZFileStreamOrientedByteWriter and ZFileStreamOrientedTextWriter are wt
debug	true or false (default is false)	Enables detailed tracing on this batch datastream.
EnablePerformanceMeasurement	true or false (default is false)	Calculates the total time spent in the batch data-streams and the processRecord method, if you are using the GenericXDBatchStep.
EnableDetailedPerformanceMeasurement	true or false (default is false)	Provides a more detailed breakdown of time spent in each method of the batch data-streams.
file.encoding	Encoding of the file.	For example, CP1047

Interface definition

```

/**
 *
 * This pattern is used to write data to z/OS dataset using
 * jzos apis
 */

```



```

public interface RecordOrientedDatasetWriterPattern {

    /**
     * This method is called during the job setup phase allowing
     * the user to do initialization.
     * The properties are the ones passed in the xJCL
     * @param props
     */
    public void initialize(Properties props);

    /**
     * This method should be used to write the given
     * object into the dataset
     * @param out
     * @param record
     * @throws IOException
     */
    public void writeRecord(ZFile out, Object record) throws IOException;

    /**
     * This method should be used to write header information
     * if any
     * @param out
     * @throws IOException
     */
    public void writeHeader(ZFile out) throws IOException;

    /**
     * This method can be optionally called during process step to explicitly
     * initialize and write the header.
     * @param header
     */
    public void writeHeader(ZFile out, Object header);

}

```

xJCL example

```

<batch-data-streams>
<bds>
<logical-name>outputStream</logical-name>
<props>
<prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.batch.samples.tests.bds.EchoWriter"/>
<prop name="DSNAME" value="USER216.BATCH.RECORD.OUTPUT"/>
<prop name="ds_parameters" value="wt"/>
<prop name="file.encoding" value="CP1047"/>
<prop name="debug" value="${debug}"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.ZFileStreamOrientedByteWriter</impl-class>
</bds>
</batch-data-streams>

```

JPAReaderPattern

The JPAReaderPattern pattern is used to retrieve data from a database using OpenJPA.

Supporting classes

The JPAReader class performs the tasks of obtaining an entity manager, running user provided queries, and iterating over the results of the query. A persistence.xml file needs to be packaged with the user application.

Required properties

The following properties are required for the pattern.

Table 63. Required properties. The table includes the name and value of each required property for the pattern.

Property name	Value
PATTERN_IMPL_CLASS	Class implementing JPAREADER Pattern interface
PERSISTENT_UNIT	The OpenJPA persistent unit name.

Optional properties

The following properties are optional for the pattern.

Table 64. Optional properties. The table includes the name, value, and description of each optional property for the pattern.

Property name	Value	Description
debug	true or false (default is false)	Enables detailed tracing on this batch data stream.
openjpa.Log	DefaultLevel=WARN,SQL=TRACE	JPA log settings
EnablePerformanceMeasurement	true or false (default is false)	Calculates the total time spent in the batch data streams and the processRecord method, if you are using the GenericXDBatchStep.
EnableDetailedPerformanceMeasurement	true or false (default is false)	Provides a more detailed breakdown of time spent in each method of the batch data streams.

Interface definition

```
public interface JPAREADERPattern {

    /**
     * This method is invoked during the job setup phase.
     *
     * @param props properties provided in the xJCL
     */

    public void initialize(Properties props);

    /**
     * This method should retrieve values for the various columns for the current row from
     * the given Iterator object. Typically this data would be used to populate an intermediate
     * object which would be returned.
     * @param listIt
     * @return
     */
    public Object fetchRecord(Iterator listIt);

    /**
     * This method should return a JPQL query that will be used during setup of the stream to
     * retrieve all relevant data that would be processed part of the job steps.
     * @return object to be used during process step.
     */
    public String getInitialLookupQuery();

    /**
     * This method gets called during Job Restart. The restart token should be used to create
     * a JPQL query that retrieves previously unprocessed records. Typically the restart token
     * is the primary key in the table and the query would get all rows with
     * primary key value > restarttoken
     * @param restartToken
     * @return The restart query
     */
    public String getRestartQuery(String restartToken);

    /**
     * This method gets called just before a checkpoint is taken.
     */
}
```

```

    * @return The method should return a string value identifying the last record read by the stream.
    */
    public String getRestartTokens();
}

```

xJCL example

```

<batch-data-streams>
<bds>
<logical-name>inputStream</logical-name>
<props>
<prop name="PERSISTENT_UNIT" value="hellojpa"/>
<prop name="debug" value="true"/>
<prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.samples.JPAInputStream"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.JPAReader</impl-class>
</bds>
</batch-data-streams>

```

JPAWriterPattern

The JPAWriterPattern pattern is used to write data to a database using a Java Persistence API (JPA) connection.

Supporting classes

The JPAWriter class implements the basic JPA operations of obtaining an EntityManager class and joining, beginning, and committing transactions. By default the JPAWriter joins an existing global transaction.

Package a persistence.xml file that sets the transaction-type attribute to JTA and declares a jta-data-source element. Optionally configure the JPAWriter class to begin and commit transactions in synchronization with the global transactions. These transactions are used with non-jta-data-source elements and connection URLs. In this case the persistence.xml file sets the transaction-type to RESOURCE_LOCAL and declare a non-jta-data-source element or connection URLs.

Required properties

The following properties are required for the pattern.

Table 65. Required properties. The table includes the name and value of each required property for the pattern.

Property name	Value
PATTERN_IMPL_CLASS	Class implementing JPAWriterPattern interface
PERSISTENT_UNIT	The OpenJPA persistent unit name
<i>JPA properties that you set on the EntityManager class</i>	<i>The value of these properties</i>

Optional properties

The following properties are optional for the pattern.

Table 66. Optional properties. The table includes the name, value, and description of each optional property for the pattern.

Property name	Value	Description
debug	true or false (The default is false.)	Enables detailed tracing on this batch data stream.
use_JTA_transactions	true or false (The default is true.)	If you use the non-jta-data-source element or connection URLs, set the value to false.
EnablePerformanceMeasurement	true or false (The default is false.)	Calculates the total time spent in the batch data-streams and the processRecord method, if you are using the GenericXDBatchStep.

Interface definition

```
public interface JPAWriterPattern {

    /**
     * This method is invoked during create job step to allow the JPAWriter stream to
     * initialize.
     * @param props Properties passed via xJCL
     */
    public void initialize(Properties props);

    /**
     * This method is invoked to actually persist the passed object to the database
     * using JPA EntityManager
     * @param manager
     * @param record
     */
    public void writeRecord(EntityManager manager, Object record);
}
```

xJCL example

```
<batch-data-streams>
<bds>
<logical-name>outputStream</logical-name>
<props>
<prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.batch.samples.tests.bds.EchoWriter"/>
<prop name="PERSISTENT_UNIT" value="mypersistentU"/>
<prop name="debug" value="true"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.JPAWriter</impl-class>
</bds>
</batch-data-streams>
```

PureQueryWriterPattern

Use this pattern to write data to a database using IBM Optim™ pureQuery Runtime. The batch data stream (BDS) framework completes the administrative tasks of opening and closing connections.

Supporting classes

The PureQueryWriter class implements the basic operations of opening and closing database connections, obtaining the PureQuery data interface, and batching operations.

Required properties

The following properties are required for the pattern.

Table 67. Required properties. The table includes the name and value of each required property for the pattern.

Property name	Value
PATTERN_IMPL_CLASS	Class implementing PureQueryWriterPattern interface
PQ_DATA_BEAN_INTERFACE	PureQuery data bean interface
ds_jndi_name	Java Naming and Directory Interface (JNDI) name of the data source to access the database

Optional properties

The following properties are optional for the pattern.

Table 68. Optional properties. The table includes the name, value, and description of each optional property for the pattern.

Property name	Value	Description
debug	true or false (The default is false.)	Enables detailed tracing on this batch data stream.
DB_SCHEMA	null	Database schema name

Table 68. Optional properties (continued). The table includes the name, value, and description of each optional property for the pattern.

Property name	Value	Description
EnablePerformanceMeasurement	true or false (The default is false.)	Calculates the total time spent in the batch data-streams and the processRecord method, if you are using the GenericXDBatchStep
force_connection_recycle	false	Forces the connection to be closed and reopened during checkpoint processing
Batch_interval	20	Number of operations to batch

Interface definition

The PureQueryWriterPattern Interface definition shows the methods that you must implement to support the PureQueryWriterPattern interface.

```
public interface PureQueryWriterPattern {

    public void initialize(Properties props);

    /**
     * The parent class passes the record to be written, the data interface, or the data interface
     * user method in order to update the database. The application might use the data interface to
     * run the pureQuery API method for in-line style or the data interface method for annotation style.
     * The parent class passes the record to be written and the
     * Data interface that may be used by the application to execute the pureQuery API method
     * (for in-line style ) or the Data interface user method ( for annotation style ) in order
     * to update the database.
     * @param
     * @param record
     * @return
     */
    public void writeRecord(Data dataInterface, Object record);
}
```

xJCL example

The example shows xJCL that you can use to define a batch data stream which implements the PureQueryWriterPattern interface in your application.

```
<batch-data-streams>
<bds>
  <logical-name>outputStream</logical-name>
  <props>
    <prop name="PATTERN-IMPL-CLASS" value="com.ibm.MyWriterPattern"/>
    <prop name="jdbc_url" value="jdbc:derby:C:\mysample\CREDITREPORT"/>
    <prop name="jdbc_driver" value="org.apache.derby.jdbc.EmbeddedDriver"/>

    <prop name="user_id" value="myid"/>
    <prop name="pswd" value="mypwd"/>
    <prop name="debug" value="true"/>
    <prop name="DB_SCHEMA" value="PQDS"/>
    <prop name="PQ_DATA_BEAN_INTERFACE" value="com.ibm..MyEmployeeData"/>
  </props>
  <impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.PureQueryWriter</impl-class>
</bds>
</batch-data-streams>
```

PureQueryReaderPattern

Use this pattern is used to read data from a database using IBM Optim pureQuery Runtime. The batch data stream (BDS) framework completes the administrative tasks of opening and closing connections.

Supporting classes

The PureQueryReader class implements the basic operations of opening and closing database connections and obtaining the IBM Optim pureQuery Runtime data.

Required properties

The following properties are required for the pattern.

Table 69. Required properties. The table includes the name and value of each required property for the pattern.

Property name	Value
PATTERN_IMPL_CLASS	Class implementing PureQueryReaderPattern interface
PQ_DATA_BEAN_INTERFACE	PureQuery data bean interface
ds_jndi_name	Java Naming and Directory Interface (JNDI) name of the data source to access the database

Optional properties

The following properties are optional for the pattern.

Table 70. Optional properties. The table includes the name, value, and description of each optional property for the pattern.

Property name	Value	Description
debug	true or false (The default is false.)	Enables detailed tracing on this batch data stream
DB_SCHEMA	null	Database schema name
EnablePerformanceMeasurement	true or false (The default is false.)	Calculates the total time spent in the batch data-streams and the processRecord method, if you are using the GenericXDBatchStep

Interface definition

The PureQueryReaderPattern Interface definition shows the methods that you must implement to support the PureQueryReaderPattern interface.

```
public interface PureQueryReaderPattern
{
    /**
     * This method is called by the batch container during step setup. The properties passed
     * in are the ones that you provide in the xJCL BDS level properties.
     * @param properties
     */
    public void initialize(Properties properties);

    /**
     * Invoked by the container during each iteration of the batch loop. This code obtains
     * the next record using the given iterator object.
     * @param iterator
     * @return
     */
    public Object fetchRecord(Iterator iterator);

    /**
     * Returns the iterator based on the passed data object that is used to iterate
     * over the records
     * @param data
     * @return
     */
    public Iterator getInitialIterator(Data data);

    /**
     * Returns the iterator based on the passed data object repositioned based on the restart
     * token of restartToken.
     * @param data
     * @param s
     * @return
     */
    public Iterator getRestartIterator(Data data, String restartToken);
}
```

```

/**
 * Invoked before a checkpoint is taken to save the restart token that is used in case
 * of a restart
 * @return
 */
public String getRestartTokens();
}

```

xJCL example

The example shows xJCL that you can use to define a batch datastream which implements the `PureQueryReaderPattern` interface in your application.

```

<batch-data-streams>
<bds>
  <logical-name>outputStream</logical-name>
  <props>
    <prop name="IMPLCLASS" value="com.ibm.MyWriterPattern"/>
      <prop name="ds_jndi_name" value="jdbc/crreport"/>
      <prop name="debug" value="true"/>
      <prop name="DB_SCHEMA" value="PQDS"/>
      <prop name="PQ_DATA_BEAN_INTERFACE" value="com.ibm.MyEmployeeData"/>
    </props>
    <impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.PureQueryReader</impl-class>
  </bds></batch-data-streams>

```

Implementing the generic batch step (GenericXDBatchStep)

A generic batch step works with one input and one output stream. This step during each iteration of the batch loop reads a single entry from the BDS Input Stream passes it to the `BatchRecordProcessor` for processing. The `BatchRecordProcessor` returns the processed data which is then passed to the BDS output stream.

About this task

Use the following properties to implement the generic batch step.

Table 71. Required properties. The table includes the property name, property value, and property description.

Property name	Value	Description
BATCHRECORDPROCESSOR	Java class name	Class implementing the <code>BatchRecordProcessor</code> interface

Table 72. Optional properties. The table includes the property name, property value, and property description.

Property	Value	Description
debug	true or false (default is false)	Enable tracing and debugging on the step
EnablePerformanceMeasurement	true or false (default is false)	Measure time spent within the step

Procedure

1. Implement the interface `com.ibm.websphere.batch.devframework.steps.technologyadapters.BatchRecordProcessor` to provide the business logic for the step. In the xJCL for the step, declare a property `BATCHRECORDPROCESSOR` with the value set to the implementation of the interface. For example:

```

...
<props>
  <prop name="BATCHRECORDPROCESSOR"
    value="com.ibm.websphere.batch.samples.tests.steps.InfrastructureVerificationTest"/>
</props>
...

```

2. Set the BDS input stream logical name to `inputStream` and a BDS output stream logical name to `outputStream`. The logical names are declared in the xJCL. For example:

```

<batch-data-streams>
  <bds>
    <logical-name>inputStream</logical-name>
    <props>
      ....
    </bds>
    <bds>
    <logical-name>outputStream</logical-name>
    <props>
      ...
    </bds>
  </batch-data-streams>

```

- While using the BatchPackager for packaging, the application for the job step class, jobstepclass, must be set to `com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep`. For example:

```

ejbname.1=IVTStep1
jndiname.1=ejb/GenericXDBatchStep
jobstepclass.1=com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep

```

Implementing the error tolerant step

An error tolerant generic batch step works with one input, one output stream, and one error stream. This step during each iteration of the batch loop reads a single entry from the batch data stream (BDS) input stream and passes it to the BatchRecordProcessor property for processing.

Before you begin

The BatchRecordProcessor property might either return a valid data object or a null value in a tolerable error. If the returned value is null, the record read from the input stream is logged on to the error stream and the `invalidRecordEncountered` method is invoked on the ThresholdPolicy interface. The threshold policy determines whether the error tolerance threshold has been reached. If so, it returns `STEP_CONTINUE_FORCE_CHECKPOINT_BEFORE_PROCESSING_CANCEL`, which forces a checkpoint and puts the job in the restartable state. Otherwise, the job continues as normal. If the data returned by `BatchRecordProcessor.processRecord` is valid, then the data is passed to the BDS output stream.

About this task

Use the following properties to implement the error tolerant step.

Table 73. Required properties. The table includes the property name, value, and description.

Property name	Value	Description
threshold_policy	Java class name	Class implementing the <code>com.ibm.websphere.batch.devframework.thresholdpolicies.ThresholdPolicy</code> interface
BATCHRECORDPROCESSOR	Java class name	Class implementing the BatchRecordProcessor interface

Table 74. Optional properties. The table includes the property name, value, and description.

Property	Value	Description
debug	true or false (default is false)	Enable tracing and debugging on the step
EnablePerformanceMeasurement	true or false (default is false)	Measure time spent within the step

Procedure

- Implement the interface `com.ibm.websphere.batch.devframework.steps.technologyadapters.BatchRecordProcessor` to provide the business logic for the step.

In the xJCL for the step, declare a property `BATCHRECORDPROCESSOR` with the value set to the implementation of the interface. For example:


```

...
<props>
  <prop name="BATCHRECORDPROCESSOR"
    value="com.ibm.websphere.batch.samples.tests.steps.InfrastructureVerificationTest"/>
</props>
...

```

2. Implement the interface `com.ibm.websphere.batch.devframework.thresholdpolicies.ThresholdPolicy` to provide the threshold policy for the step. You can also use the product implementations such as `com.ibm.websphere.batch.devframework.thresholdpolicies.PercentageBasedThresholdPolicy` or `com.ibm.websphere.batch.devframework.thresholdpolicies.RecordBasedThresholdPolicy`.

Declare the `ThresholdPolicy` to use in the xJCL as shown in the following code snippet:

```

...
<props>
  <prop name="threshold_policy"
    value="com.ibm.websphere.batch.devframework.thresholdpolicies.PercentageBasedThresholdPolicy"/>
</props>
...

```

3. Set the BDS input stream logical name to `inputStream` and a BDS output stream logical name to `outputStream` and the BDS output stream for errors to `errorStream`.

The logical names are declared in the xJCL. For example:

```

<batch-data-streams>
  <bds>
    <logical-name>inputStream</logical-name>
  </bds>
  <bds>
    <logical-name>outputStream</logical-name>
  </bds>
  <bds>
    <logical-name>errorStream</logical-name>
  </bds>
</batch-data-streams>

```

4. While using the `BatchPackager` for packaging, the application for the job step class, `jobstepclass`, must be set to `com.ibm.websphere.batch.devframework.steps.technologyadapters.ThresholdBatchStep`. For example:

```

ejbname.1=IVTStep1
jndiname.1=ejb/MyThresholdBatchStep
jobstepclass.1=com.ibm.websphere.batch.devframework.steps.technologyadapters.ThresholdBatchStep

```

Declaring the percentage-based threshold policy (PercentageBasedThresholdPolicy)

The `PercentageBasedThresholdPolicy` policy provides a batch implementation of the `ThresholdPolicy` interface.

Before you begin

The `percentageBasedThresholdPolicy` policy applies only if the `ThresholdBatchStep` is used. It calculates the percentage of the number of error records processed to the total number processed. If the result is greater than the threshold, it forces the job to go into restartable state.

About this task

Optionally use the following properties when you declare the percentage-based threshold policy.

Table 75. Optional properties. The table includes the property name, property value, and property description.

Property	Value	Description
debug	true or false (default is false)	Enable tracing and debugging on the step
minimum_threshold_sample_size	Integer value (default is 20)	The minimum number of records to process before checking for threshold breach.
threshold_threshold	Double value (default is 0.1)	The acceptable percentage of errors.

Procedure

Declare the threshold policy to use in the xJCL as a property of the step as follows:

```
...
<props>
  <prop name="threshold_policy"
    value="com.ibm.websphere.batch.devframework.thresholdpolicies.PercentageBasedThresholdPolicy"/>
</props>
...
```

Declaring the record based threshold policy (RecordBasedThresholdPolicy)

The RecordBasedThresholdPolicy policy provides a batch implementation of the ThresholdPolicy interface.

Before you begin

The record based threshold policy of RecordBasedThresholdPolicy is applicable only if the threshold batch step of ThresholdBatchStep is used. It counts the number of error records processed. If the result is greater than the threshold, it forces the job to go into restartable state.

About this task

Optionally use the following properties when you declare the record based threshold policy.

Table 76. Optional properties. The table includes the property name, property value, and property description.

Property	Value	Description
debug	true or false (default is false)	Enable tracing and debugging on the step
minimum_threshold_size	Integer value (default is 20)	The minimum number of records to process before checking for threshold breach.
error_threshold	Double value (default is 100)	The number of error records.

Procedure

Declare the threshold policy to use in the xJCL as a property of the step as follows:

```
...
<props>
  <prop name="threshold_policy"
    value="com.ibm.websphere.batch.devframework.thresholdpolicies.RecordBasedThresholdPolicy"/>
</props>
...
```

Chapter 5. Deploying batch applications

This section covers such areas as packaging Enterprise JavaBeans (EJB) 3.0 and later modules and installing batch applications.

Packaging EJB modules in a batch application using Rational Application Developer

Use Rational Application Developer 8.0.1 or later to package Enterprise JavaBeans (EJB) 3.0 and later modules.

Before you begin

Develop a batch application.

Procedure

1. Import your batch application EAR file into Rational Application Developer. Click **File > Import**.
 - a. Select **Java EE - EAR file**. Click **Next**.
 - b. Enter the location of your EAR file in the EAR file field.
 - c. Set the Target run time to a WebSphere Application Server Version 8.5 run time.
 - d. Clear everything on the **EAR Module and Utility JAR Projects** page (third page).
 - e. Click **Finish**.
2. Link in your EJB 3.0 or later JAR file.
 - a. Right-click on the newly created EAR project.
 - b. Select **Properties > Java EE Module Dependencies**
 - c. Click **Add External JARs...**
 - d. Select your EJB JAR file, such as an EJB3 JAR.
 - e. Click **OK**.
3. Export your EAR file.
 - a. Right-click on the EAR project.
 - b. Click **Export > EAR file**.
 - c. Enter a value in the **Destination** field.
 - d. Click **Finish**.

Installing the batch application

Install a batch application the same way that you install an enterprise application.

Before you begin

Develop a compute-intensive application or a transactional batch application. You can develop a compute intensive application using a compute-intensive job controller, the command line, or the Apache ANT tool. You can develop a transactional batch application using a batch job controller and Enterprise JavaBeans (EJB) data stream, the command line, or the ANT tool.

About this task

A batch application is installed like Java Platform, Enterprise Edition (Java EE) transactional applications.

When mapping modules of the batch application to servers, select the server or cluster that you created previously for the enterprise bean modules that contain the logic for a batch job.

Procedure

1. Install the batch application using the administrative console, using wsadmin scripting, or using another supported way to install enterprise applications.

See topics on installing enterprise application files.

2. Verify that the application installed correctly.

For example, go to the Enterprise applications administrative console page by clicking **Applications > Application Types > WebSphere enterprise applications**. If the application is not running, select the application and click **Start**. Test the application to ensure that it operates correctly.

What to do next

Configure the job scheduler and submit batch jobs using the job scheduler.

Deploying an OSGi batch application

You can package an existing batch application as an OSGi application. You then deploy the package so that you can expose batch artifacts as services, making those artifacts visible to the batch container.

About this task

Package an OSGi batch application, modify the Blueprint xml file to describe batch artifacts as services, and export the OSGi batch application as an enterprise bundle archive (EBA). Then create the xJCL. Finally, deploy the OSGi batch application.

Procedure

1. Package an OSGi batch application.

You package the OSGi batch application as an EBA. The EBA contains at least your batch bundle, which is a Blueprint bundle. The API bundle and the dispatcher bundles are installed once into the internal bundle repository.

Read the topic on creating a client bundle and follow the steps to package your OSGi application.

2. Author the Blueprint xml.

You must declare the job steps and batch data streams as services so that the scheduler can invoke them. If your OSGi batch application implements a checkpoint policy algorithm or a results algorithm, then you must also declare as a service each algorithm that the application implements.

- a. Declare each job step as a Blueprint service.

- 1) Set the interface attribute.

a) If the step is a compute intensive step, set the attribute to `com.ibm.websphere.ci.CIWork`.

b) If the step is a transactional batch step, set the attribute to `com.ibm.websphere.batch.BatchJobStepInterface`.

- 2) Set the ref attribute to the bean ID that declares the step bean.

3) Declare a property with the `xjcl:classname` key and a value that is the Java class that implements the step.

4) Declare a bean for the Java class that implements the step.

5) Set the scope attribute to `prototype`.

Compute intensive step example:

```
<bean id="IVTStep1" class="com.ibm.websphere.batch.samples.tests.steps.GenerateDataStep" scope="prototype"/>
```

```
<service ref="IVTStep1" interface="com.ibm.websphere.ci.CIWork" id="step1">
```

```

<service-properties>
  <entry key="xjcl:classname" value="com.ibm.websphere.batch.samples.tests.steps.GenerateDataStep"/>
</service-properties>
</service>

```

Transactional batch step example:

```

<bean id="EchoStep2" class="com.ibm.websphere.batch.samples.tests.steps.TestBatchJobStep" scope="prototype"/>
<service ref="EchoStep2" interface="com.ibm.websphere.batch.BatchJobStepInterface" id="echostep1">
  <service-properties>
    <entry key="xjcl:classname" value="com.ibm.websphere.batch.samples.tests.steps.TestBatchJobStep"/>
  </service-properties>
</service>

```

b. Declare each batch data stream as a Blueprint service.

- 1) Set the interface attribute to `com.ibm.websphere.batch.BatchDataStream`.
- 2) Set the ref attribute to the bean ID that declares the batch data stream bean.
- 3) Declare a property with the `xjcl:classname` key and a value that is the Java class that implements the batch data stream.
- 4) Declare a bean for the Java class that implements the batch data stream.
- 5) Set the scope attribute to `prototype`.

Batch data stream example:

```

<bean id="output" class="com.ibm.websphere.batch.samples.tests.bds.TestOutputBatchDataStream" scope="prototype"/>
<service ref="output" interface="com.ibm.websphere.batch.BatchDataStream" id="out1">
  <service-properties>
    <entry key="xjcl:impl-class" value="com.ibm.websphere.batch.samples.tests.bds.TestOutputBatchDataStream"/>
  </service-properties>
</service>

```

c. Declare the checkpoint policy algorithm as a Blueprint service.

If your OSGi batch application implements a checkpoint policy algorithm, then declare the algorithm as a Blueprint service. Otherwise, skip this step.

- 1) Set the interface attribute to `com.ibm.wsspi.batch.CheckpointPolicyAlgorithm`.
- 2) Set the ref attribute to the bean ID that declares the checkpoint bean.
- 3) Declare a property with the `xjcl:classname` key and a value that is the Java class that implements the checkpoint policy algorithm.
- 4) Declare a bean for the Java class that implements the checkpoint policy algorithm.
- 5) Set the scope attribute to `prototype`.

Checkpoint policy algorithm example:

```

<bean id="chkpt" class="com.ibm.websphere.batch.samples.MyCheckpointAlgorithm" scope="prototype"/>
<service ref="chkpt" interface="com.ibm.wsspi.batch.CheckpointPolicyAlgorithm" id="ck1">
  <service-properties>
    <entry key="xjcl:impl-class" value="com.ibm.websphere.batch.samples.MyCheckpointAlgorithm"/>
  </service-properties>
</service>

```

d. Declare the results algorithm as a Blueprint service.

If your OSGi batch application implements a results algorithm, then declare the algorithm as a Blueprint service. Otherwise, skip this step.

- 1) Set the interface attribute to `com.ibm.wsspi.batch.ResultsAlgorithm`.
- 2) Set the ref attribute to the bean ID that declares the results algorithm bean.
- 3) Declare a property with the `xjcl:classname` key and a value that is the Java class that implements the results algorithm.
- 4) Declare a bean for the Java class that implements the results algorithm.
- 5) Set the scope attribute to `prototype`.

Results algorithm example:

```

<bean id="myres" class="com.ibm.websphere.batch.samples.MyResultsAlgorithm" scope="prototype"/>
<service ref="myres" interface="com.ibm.wsspi.batch.ResultsAlgorithm" id="r1">
  <service-properties>
    <entry key="xjcl:impl-class" value="com.ibm.websphere.batch.samples.MyResultsAlgorithm"/>
  </service-properties>
</service>

```

3. Export the OSGi batch application as an EBA.

4. Create the xJCL.

Create the xJCL as you would for other batch applications, with a few differences:

- Make the application-name attribute on the job step the deployed asset name. The deployed asset is a composition unit.
- Make the classname subelement of the step match the xjcl:classname property of the step service. The following example uses the xjcl:classname property of `com.ibm.websphere.batch.samples.tests.steps.TestBatchJobStep` from the transactional batch step example listed previously in this procedure.

```
<step id="step1">  
<classname> com.ibm.websphere.batch.samples.tests.steps.TestBatchJobStep</classname>  
</step>
```

5. Deploy the OSGi batch application.

Read the topic on deploying an OSGi application as a business-level application and follow the steps.

Results

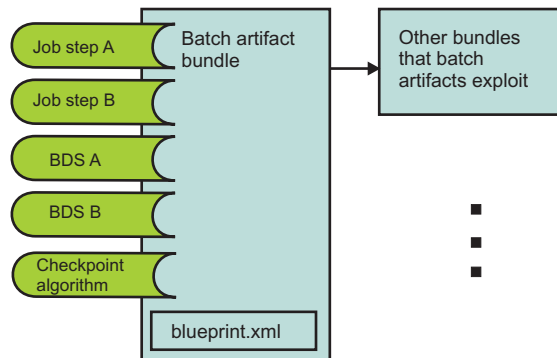
You packaged an OSGi batch application, modified the Blueprint xml to describe batch artifacts as services, and exported the OSGi batch application as an enterprise bundle archive (EBA). Then you created the xJCL. Finally, you deployed the OSGi batch application.

OSGi batch applications

OSGi batch applications are batch applications that you can package and deploy as OSGi applications so that you can expose batch artifacts as services.

An OSGi batch application has characteristics of both batch applications and OSGi applications. The batch application characteristics include artifacts such as job steps and batch data streams. The OSGi batch application is like an OSGi application because it exposes the batch artifacts as services through a Blueprint bundle so that the batch artifacts are visible to the batch container.

The following diagram shows the bundle organization of a batch application in the batch container. The batch artifact bundle contains the components of multiple job steps, multiple batch data streams, and a checkpoint algorithm. It also contains a Blueprint XML file that defines and describes the assembly of the components. The diagram shows that other bundles can exist in the batch container.



Submitting batch jobs

You can submit batch applications using the job scheduler Enterprise JavaBeans (EJB) interface or the job scheduler web services interface.

xJCL elements

Jobs are expressed using an Extensible Markup Language XML dialect called xJCL (XML Job Control Language). This dialect has constructs for expressing all of the information needed for both compute-intensive and batch jobs, although some elements of xJCL are only applicable to compute-intensive or batch jobs. See the xJCL provided with the Sample applications, the xJCL table, and xJCL XSD schema document for more information about xJCL. The xJCL definition of a job is not part of the batch application, but is constructed separately and submitted to the job scheduler for to run. The job scheduler uses information in the xJCL to determine where and when to run the job.

xJCL elements

The following table summarizes the xJCL elements.

Table 77. xJCL elements. The table includes xJCL elements, whether each xJCL element applies to Java Platform, Enterprise Edition (Java EE) compute-intensive or batch jobs, and subelements, attributes, and descriptions for each xJCL element.

Element	Java EE compute-intensive	Java EE Batch	Subelement	Attributes	Description
job	yes	yes			Scopes the description of a batch job.
job	yes	yes		name	Name of the job. This name must match the name of the batch application
job	yes	yes		accounting	Optional accounting information attribute.
job	yes	yes		class	Optional job class attribute, which identifies the job class under which the job runs.
job	yes	yes		default-application-name	The application name to be used when no job step application-name attribute is found. The application name to be used. For OSGi batch applications, format the name as <code>osgi:<eba name>:<version></code> .
job	yes	yes	jndi-name		JNDI name that is given to the job controller stateless session bean when the batch application is deployed into the product.
job	yes	yes	job-scheduling-criteria	required-capability	The required-capability of the job, which must be defined on an endpoint for the job to be dispatched to that endpoint.
job	yes	yes	step-scheduling-criteria	See step-scheduling-criteria element	
job	no	yes	checkpoint-algorithm	See checkpoint-algorithm element	
job	no	yes	results-algorithm	See results-algorithms element	
job	yes	yes	substitution-props++	See prop element	The required-capability of the job, which must be defined on an endpoint for the job to be dispatched to that endpoint
job-step	yes*	yes		name	Optional name of the step. This information is returned on operational commands.
job-step	yes*	yes		application-name	Optional name of the application run by the step. The attribute name is used if application-name is omitted and the job level attribute default-application-name is omitted.
job-step	no	yes	step-scheduling	See step-scheduling element	Allows for conditional logic based on return codes of steps that determines whether the batch step is invoked.
job-step	yes	yes	classname		Fully-qualified name of class that implements the compute intensive job.
job-step	yes	no	checkpoint-algorithm-ref		Specifies the checkpoint algorithm to use for the batch job step.
job-step	no	yes	results-ref	See results-ref element	Specifies the results algorithm to use for the conditional batch job step execution.
job-step	no	yes	batch-data-streams	See batch-data-streams element	A sequence of bds elements. Each bds is the configuration information necessary to create a batch data stream.

Table 77. xJCL elements (continued). The table includes xJCL elements, whether each xJCL element applies to Java Platform, Enterprise Edition (Java EE) compute-intensive or batch jobs, and subelements, attributes, and descriptions for each xJCL element.

Element	Java EE compute-intensive	Java EE Batch	Subelement	Attributes	Description
job-step	yes	yes	props	See props element	Name-value properties to pass to the step.
job-step	no	no	exec	See exec element	Identifies the executable associated with the job step.
job-step	no	no	env-entries	See env-entries element	Identifies the environmental properties associated with the job step.
prop	yes	yes			Single instance of a name value pair, that serves as a property.
prop	yes	yes		name	Name of the property.
prop	yes	yes		value	Value of the property.
props	yes	yes	prop	See prop element	
env-entries	no	no			Series of prop elements that are used to pass name-value pair properties to steps, bds, checkpoint algorithms, and results algorithms.
env-entries	no	no	env-var	See env-var element	
exec	no	no			Series of prop elements that are used to pass name-value pair properties to steps, bds, checkpoint algorithms, and results algorithms.
exec	no	no		executable	The name of the executable associated with the job step.
exec	no	no	arg	See line element	
line	no	no			Command-line arguments passed to the job step executable.
bds	no	yes			Single instance of a batch data stream implementation made available to the batch job.
bds	no	yes	logical-name		A string that is embedded in batch step, which uses it to query the batch runtime environment for a specific batch data stream instance.
bds	no	yes	impl-class		Fully-qualified class name of the batch data stream implementation class.
bds	no	yes	props	See props elements	List of properties that are passed to the batch data stream implementation class.
batch-data-streams	no	yes			Series of bds elements
batch-data-streams	no	yes	bds	See bds element	
step-scheduling	no	yes			Applies to job-steps to create return code-based conditional flows for a batch job. Compares values of return codes defined for this batch job to decide whether a step is invoked or not while processing a batch job. The values of return codes are compared using the returncode-expression element.
step-scheduling	no	yes	returncode-expression	see returncode-expression	Returncode- expression to evaluate.
step-scheduling	no	yes		condition	If there is more than one returncode-expression element in the step-scheduling element, conditional operators are applied to them. Conditional operators supported are: AND, OR.
returncode-expression	no	yes			Used under step-scheduling tags to decide whether a batch job step runs based on return codes of other job steps.
returncode-expression	no	yes		step	Name of step whose return code is to be compared in this expression.
returncode-expression	no	yes		operator	Operator to use for the return code expression. The supported operators are eq for equals, lt for less than, gt for greater than, le for less than or equal to, and ge for greater than or equal to.
returncode-expression	no	yes		value	The value with which to compare the return code.
step-scheduling-criteria	no	yes			Describes the sequence in which the job steps are processed. Currently sequential scheduling is supported; for example, steps get invoked in the order in which they exist in xJCL.

Table 77. xJCL elements (continued). The table includes xJCL elements, whether each xJCL element applies to Java Platform, Enterprise Edition (Java EE) compute-intensive or batch jobs, and subelements, attributes, and descriptions for each xJCL element.

Element	Java EE compute-intensive	Java EE Batch	Subelement	Attributes	Description
step-scheduling-criteria	no	yes	scheduling-mode		Sequence in which to invoke steps, only possible value is sequential right now.
checkpoint-algorithm	no	yes			Declares a checkpoint algorithm that can be used for a batch job step.
checkpoint-algorithm	no	yes		name	Name of algorithm.
checkpoint-algorithm	no	yes	classname		Class that implements this algorithm.
checkpoint-algorithm	no	yes	props	See props element	Sequence of prop elements for the checkpoint algorithm.
checkpoint-algorithm-ref	no	yes			Reference to a checkpoint algorithm element.
checkpoint-algorithm-ref	no	yes		name	Name of checkpoint algorithm to which you are referring.
checkpoint-algorithm-ref	no	yes	props	See props element.	Sequence of prop elements for the checkpoint algorithm.

++ The xJCL element substitution-props is discussed in the following section.

xJCL substitution-props

The job xJCL can contain symbolic variables. A symbolic variable is an expression of the form `${variable-name}`, which is found outside a comment in an otherwise well-formed document. For example:

```
<checkpoint-algorithm-ref name="${checkpoint}" />
```

The xJCL element, substitution-props, defines a default name and value pairs for symbolic variables. Following is an example of the substitution-props element, taken from the postingSampleXJCL.xml document:

```
<substitution-props>
<prop name="wsbatch.count" value="5" />
<prop name="checkpoint" value="timebased" />
<prop name="checkpointInterval" value="15" />
<prop name="postingsDataStream" value="${was.install.root}${file.separator}temp${file.separator}postings" />
</substitution-props>
```

Substitution for symbolic variables occurs at run time. At run time, the string `${variable-name}` is replaced with the value of the property when the xJCL is submitted for execution. Using the properties in the previous example, the string `${checkpoint}` is replaced with the string time-based before the job is submitted.

Symbolic variables can be indirect. For example: `name=FILENAME value=${${filename}}` used with the name/value pair `filename=postingsDataStream` yields the same result as specifying `name=FILENAME value=${postingsDataStream}`.

Symbolic variables can also be compound. For example, `name=postingsDataStream value=${was.install.root}${file.separator}temp${file.separator}postings`.

The name/value pairs do not have to be defined in the job document substitution-props element. The props name and value pairs defined in the substitution-props element are default values for the named variables. If not defined in the substitution-props element, name/value pairs must be either passed in via

the job scheduler APIs when the job is submitted or defined in the system properties for the JVM. Every symbolic variable defined in the body of a job document must be resolved for the xJCL to be considered valid. Every name/value pair defined in the job document must resolve to a symbolic variable which is found in the body of the xJCL for the xJCL to be considered valid.

If name/value pairs are both defined in the xJCL document and passed to the job scheduler APIs at job submission time, the name/value pairs passed via the Job Scheduler APIs override the default values defined in the xJCL document. If name/value pairs are neither passed in via the job scheduler APIs nor defined as defaults in the xJCL document, name/value pairs for the symbolic variables must be defined in the system JVM properties for the xJCL to be considered valid.

Symbolic variables are resolved by the job scheduler before job submission, except for the following special variables, which are resolved at the grid endpoint. The following special variables all must be defined as JVM system properties. They are:

- `${was.install.root}`
- `${user.install.root}`
- `${agent.home}`

Batch job state table

As the job scheduler and grid endpoint process a batch job, the job state updates in the job scheduler database. The diagram shows the relationship between states, and the following table lists the possible batch job states and the events that trigger transitions between states. You can view the current state of a batch job from the job management console, or retrieve it using the command line or Enterprise JavaBeans (EJB) interface. If a failure occurs before a batch step initializes, then the batch job goes into execution failed state. Otherwise, it goes into restartable state.

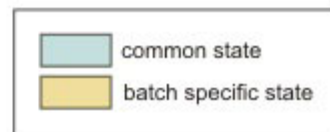
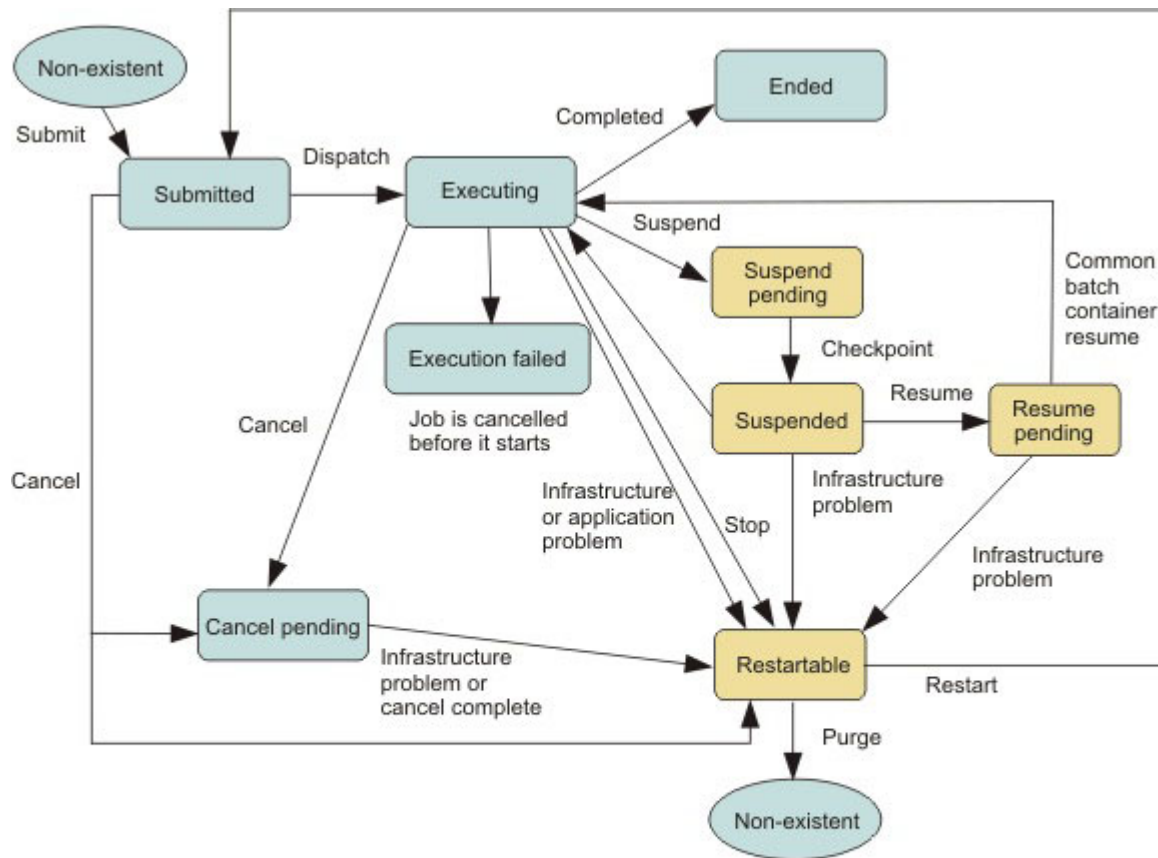


Table 78. Batch job states. The table includes each batch start state with its client command, system action, special condition, numeric return code, and end state. An empty table cell indicates that there is not a client command, system action, condition, or return code for the start state.

Start state	Client command	System action	Special condition	Return code	End state
non-existent (delayed submit)	submit				pending submit
non-existent	submit				submitted
submitted		dispatch		0	executing
submitted	cancel			0	restartable
executing	stop			0	restartable
executing	cancel			4	cancel_pending
executing		caught application error*		4	restartable
executing			Infrastructure problem**	4	restartable/unknown
executing	suspend			4	suspend_pending
executing		job completed		4	ended
executing			Infrastructure problem in job setup***	4	restartable
suspend_pending		checkpoint		2	suspended
suspend_pending			Infrastructure problem**	2	restartable/unknown
suspended	resume			5	resume_pending

Table 78. Batch job states (continued). The table includes each batch start state with its client command, system action, special condition, numeric return code, and end state. An empty table cell indicates that there is not a client command, system action, condition, or return code for the start state.

Start state	Client command	System action	Special condition	Return code	End state
suspended	cancel			5	cancel_pending
suspended			Infrastructure problem**	5	restartable/unknown
resume_pending		job resumed		2	executing
resume_pending			Infrastructure problem**	2	restartable/unknown
restartable	restart			8	submitted
cancel_pending		job canceled		1	restartable
cancel_pending			Infrastructure problem**	1	restartable/unknown
restartable	purge			8	non-existent
execution_failed	purge			9	non-existent
ended	purge			7	non-existent

Table 79. Notes for the batch job states table. The table includes each note with a description.

Note	Description
* Application error	The batch application failed at run time. The grid endpoints detected this failure.
** Infrastructure problem	An unexpected error has occurred. See the following example for infrastructure problem in job setup.
*** Infrastructure problem in job setup	<p>An unexpected error that occurs when a batch job is set up for the first time by the grid endpoints. For example, if there is an unexpected database failure, the job goes into execution_failed state.</p> <ul style="list-style-type: none"> In this condition, the batch job is run for the first time and no steps are processed yet. Batch jobs go into the restartable state under most failure conditions so that they can restart from checkpointed positions if the failure condition can be overcome. However, in this instance of a failure condition, a batch job goes into execution_failed state and cannot be restarted. Since this situation is a job setup scenario and work is not yet processed by the batch job, batch work is not lost as a result of failure. If jobs are in a non-final state on the endpoint, the scheduler puts the jobs into an unknown state under two conditions. The conditions are that the endpoint loses communications or the endpoint goes down. If the endpoint comes back up, the scheduler synchronizes the job status with the endpoint. If the endpoint goes down, all batch jobs are put into a restartable state and all compute-intensive jobs in an execution failed state. If the endpoint has only lost communication with the scheduler and the jobs continue to run, the scheduler updates its status. The status update is the final state of the jobs running on the endpoint at that point.

Native execution job state table

As the job scheduler and grid endpoint process a native execution job, the job state updates in the job scheduler database.

The following table lists the possible states that a native execution job can have and the events that trigger state transitions. You can view the current state of a native execution job in the administrative console job management pages. You can retrieve the state using the command line, enterprise bean, or web service interfaces to the job scheduler.

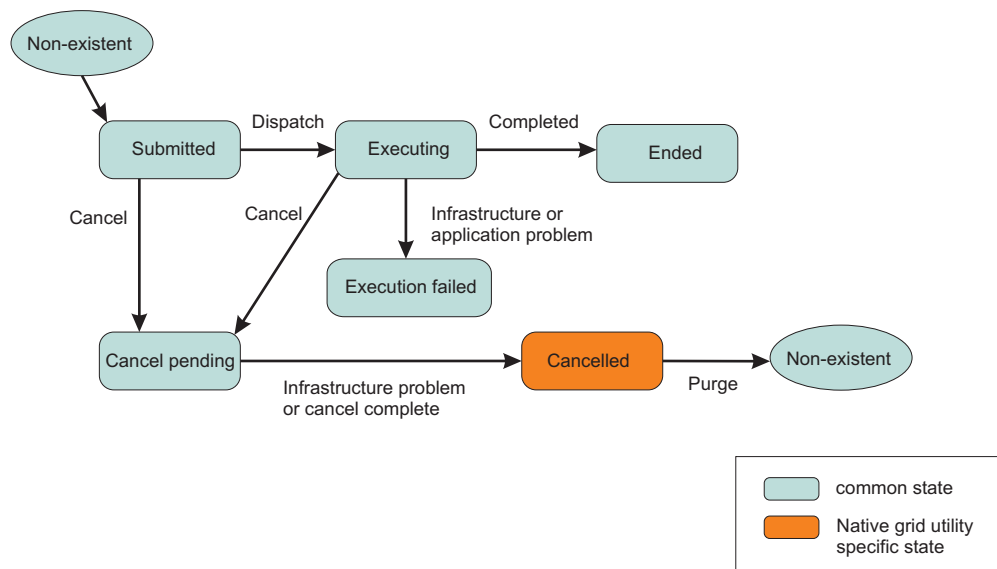


Table 80. Native grid utility job state table. The table lists the start state, the client command, the grid action, the special condition, and the end state. An empty table cell indicates that there is not a client command, action, condition, or end state for the start state.

Start state	Client command	Grid action	Special condition	End state
non-existent	submit	submitted		
submitted		dispatch		executing
submitted	cancel			canceled
executing		job completed		ended
executing	cancel			cancel-pending
cancel_pending		job canceled		canceled
cancel_pending			Infrastructure problem	execution_failed
canceled	purge			non-existent
execution_failed	purge			non-existent
ended	purge			non-existent

Submitting batch jobs using the job scheduler EJB interface

The job scheduler Enterprise JavaBeans (EJB) interface is used to programmatically submit and manipulate a batch job. You can use the EJB interface with the base scheduler in WebSphere Application Server to perform calendar-based submission of a batch job.

Before you begin

The job scheduler supports programmatic access to its functions over both an EJB interface for Java Platform, Enterprise Edition (Java EE) applications and a web services interface for both Java EE and non-Java EE applications. The EJB interface for the job scheduler is described by the interfaces found in the API documentation. Consult this documentation for further information.

Develop and install your batch applications.

About this task

This topic describes how to submit a batch job to the job scheduler using the base scheduler. It includes a code example that demonstrates how to invoke the job scheduler EJB.

Procedure

1. Create and configure a scheduler. Read about how to create and configure a scheduler in the topic on developing and scheduling tasks.
2. Create a scheduler task for submitting batch work.

This scheduler task invokes the job scheduler EJB to submit a batch job. Read the instructions for creating a task that invokes an EJB in the topic on developing a task that calls a session bean. This topic also includes instructions for using the calendaring feature of the scheduler. The following example demonstrates on how to invoke the job scheduler EJB:

```
// These are the import statements needed by the task
import javax.naming.*;

import com.ibm.websphere.longrun.JobScheduler;
import com.ibm.websphere.longrun.JobSchedulerHome

private JobSchedulerHome zjsHome = null;
private JobScheduler zjs = null;

public void process(TaskStatus task) ()
try{

    //Ensure that the xJCL can be placed in a string, for example, by reading an xJCL
    //File into a string
    String xJCL = <xJCL as a string>;

    //Obtain cell-level naming context
    InitialContext ctxt = new InitialContext();
    Hashtable env = new Hashtable();

    env.put (Context.INITIAL_CONTEXT_FACTORY,
            "com.ibm.websphere.naming.WsnInitialContextFactory");

    env.put(Context.PROVIDER_URL,"corbaloc:rir:/NameServiceCellRoot");
    ctxt = new InitialContext(env);

    //To look up the LRS EJB from the cell context in the namespace,
    //The name context to the application server or cluster to which the LRS
    //Application is deployed has to be provided
    //Eg: "nodes/myNode/servers/myServer" or "clusters/myCluster".

    String longRunningContext = <long_running_context>;

    zjsHome = (JobSchedulerHome) ctxt.lookup(longRunningContext +
            "/ejb/com/ibm/websphere/longrun/JobSchedulerHome");
    zjs = zjsHome.create();
    zjs.submitJob( xJCL );

}catch (Exception e) {
    System.out.println(e.getMessage());
}
```

3. Run the program to submit batch work.

Read the topic on submitting a task to a scheduler.

Job scheduler EJB interfaces

The job scheduler Enterprise JavaBeans (EJB) interface is used to programmatically submit a batch job to the job scheduler and manipulate the job.

The following code is the remote interface to the job scheduler EJB. The code produces application programming interfaces.

Some code is split on multiple lines for printing purposes.

```
/**
 * This is the remote interface for the Job Scheduler EJB.
 * Clients of this interface can programmatically submit and manipulate jobs to the
```

```

* Job Scheduler. Code similar to the following can be used to lookup and invoke
* the remote Job Scheduler EJB interface:
*
* InitialContext ctxt = new InitialContext();
* Hashtable env = new Hashtable();
*
* env.put (Context.INITIAL_CONTEXT_FACTORY,
* "com.ibm.websphere.naming.WsnInitialContextFactory");
* env.put(Context.PROVIDER_URL,
* "corbaloc:iiop:<schedulerHostName>:<schedulerBootstrapPort>/NameServiceCellRoot");
* ctxt = new InitialContext(env);
*
* // In order to lookup the Job Scheduler EJB from the cell context in the namespace,
* // the name context to the application server or cluster to which the Job Scheduler
* // application is deployed has to be provided.
* // Eg: "nodes/myNode/servers/myServer" or "clusters/myCluster".
*
* String jobSchedulerContext = clusters/myCluster;
*
* JobSchedulerHome zjsHome = (JobSchedulerHome)
*     PortableRemoteObject.narrow(ctxt.lookup(jobSchedulerContext +
*         "/ejb/com/ibm/websphere/longrun/JobSchedulerHome"),JobSchedulerHome.class);
*
* JobScheduler js = zjsHome.create();
*
*
* @ibm-api
*/
public interface JobScheduler extends javax.ejb.EJBObject {

    /**
     * Submits the specified job, saved in the xJCL repository, to the job scheduler
     * @param job The name of the job that was stored to the xJCL repository
     * @return the job ID assigned by the job scheduler to the submitted job
     *
     * @throws InvalidJobNameException if job is not found in the xJCL repository.
     * @throws SchedulerException if an unexpected error is thrown by the
     * job scheduler while submitting the job
     * @throws JCLEException if the xJCL stored in the repository is corrupted or not valid.
     * @throws JobSubmissionException if an error occurs while submitting the job
     * @throws java.rmi.RemoteException
     */
    public String submitJobFromRepository( String job ) throws
        InvalidJobNameException,
        SchedulerException,
        JCLEException,
        JobSubmissionException,
        java.rmi.RemoteException;

    /**
     * Submits the job, which is defined by the xJCL, to the job scheduler
     *
     * @param xJCL The xJCL for the job
     * @return the job ID assigned by the job scheduler to the submitted job
     *
     * @throws SchedulerException if an unexpected error is thrown by the
     * job scheduler while submitting the job
     * @throws JCLEException if the xJCL stored in the repository is corrupted or not valid.
     * @throws JobSubmissionException if an error occurs while submitting the job
     * @throws java.rmi.RemoteException
     */
    public String submitJob( String xJCL ) throws
        SchedulerException,
        JCLEException,
        JobSubmissionException,
        java.rmi.RemoteException;

    /**
     * Submits the job specified by the xJCL passed in to the job scheduler and

```

```

* saves the xJCL to the xJCL repository.
*
* @param xJCL The xJCL for the job
* @param job The name given to the saved job in the xJCL repository.
* This name can be used when invoking the submitJobFromRepository
* method.
* @param replace A boolean indicating if the xJCL in the repository should
* be replaced, in case a job by that name already exists
* in the xJCL repository.
*
* @return the job ID assigned by the job scheduler to the submitted job
*
* @throws InvalidOperationException if the job already exists in the xJCL repository
* and the replace parameter specified is false
* @throws SchedulerException if an unexpected error is thrown by the job scheduler
* while submitting the job
* @throws JCLException if the xJCL stored in the repository is corrupted or not valid.
* @throws JobSubmissionException if an error occurs while submitting the job
* @throws java.rmi.RemoteException
*
*/
public String saveJobToRepositoryAndSubmit( String xJCL, String job, boolean replace ) throws
    InvalidOperationException,
    SchedulerException,
    JCLException,
    JobSubmissionException,
    java.rmi.RemoteException;

/**
* Purges the job, identified by the job ID, from the job scheduler and the grid endpoint
* environments.
*
* @throws InvalidJobIDException if no job by the specified job ID exists in the job scheduler
* @throws SchedulerException if an unexpected error is thrown by the job scheduler while
* purging the job
* @throws java.rmi.RemoteException
*
* @param jobid The ID of the job to be purged
*/
public void purgeJob( String jobid ) throws
    InvalidJobIDException,
    SchedulerException,
    java.rmi.RemoteException;

/**
* Cancels the job identified by the job ID
*
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws InvalidJobIDException if no job by the specified job id exists in the
* job scheduler
* @throws SchedulerException if an unexpected error is thrown by the job scheduler
* while canceling the job
* @throws java.rmi.RemoteException
*
* @param jobid The ID of the job
*/
public void cancelJob( String jobid ) throws
    InvalidOperationException,
    InvalidJobIDException,
    SchedulerException,
    java.rmi.RemoteException;

/**
* Forcibly cancels the job identified by the job ID
*
* Supported on z/OS only. The forcedCancelJob request will be processed as a
* cancelJob request on distributed platforms.
*
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws InvalidJobIDException if no job by the specified job ID exists in the

```



```

* job scheduler
* @throws SchedulerException if an unexpected error is thrown by the job scheduler
* while canceling the job
* @throws java.rmi.RemoteException
*
* @param jobid The ID of the job
*/
public void forcedCancelJob( String jobid ) throws
    InvalidOperationException,
    InvalidJobIDException,
    SchedulerException,
    java.rmi.RemoteException;

/**
* Restarts the job identified by the job ID. Only jobs in the restartable state can be
* restarted.
*
* @throws InvalidJobIDException if no job by the specified job ID exists in the
* job scheduler
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws SchedulerException if an unexpected error is thrown by the job scheduler while
* restarting the job
* @throws JCLException if the xJCL for the job is corrupted or not valid.
* @throws JobSubmissionException if an error occurs while submitting the job
* @throws java.rmi.RemoteException
*
* @param jobid The ID of the job
*/
public void restartJob( String jobid ) throws
    InvalidJobIDException,
    InvalidOperationException,
    SchedulerException,
    JCLException,
    JobSubmissionException,
    java.rmi.RemoteException;

/**
* Returns the job status for the given job ID. Refer to {@link JobStatusConstants
* JobStatusConstants} for a
* list of the job status codes returned by this method.
*
* @param jobid The ID of the job
*
* @throws InvalidJobIDException if no job by the specified job ID exists in the
* job scheduler
* @throws SchedulerException if an unexpected error is thrown by the job scheduler
* while processing the command
* @throws java.rmi.RemoteException
*
* @return the status of the job
*/
public int getJobStatus( String jobid ) throws
    InvalidJobIDException,
    SchedulerException,
    java.rmi.RemoteException;

/**
* Returns the job output for a given job ID that displays the job's progress. This only
* applies to batch jobs.
*
* @param jobid The ID of the job
*
* @throws InvalidJobIDException if no job by the specified job ID exists in the job scheduler
* @throws SchedulerException if an unexpected error is thrown by the job scheduler while
* processing the command
* @throws java.rmi.RemoteException
*
* @return the job output of the job
*/
public String getJobOutput( String jobid ) throws
    InvalidJobIDException,
    SchedulerException,

```

```

        java.rmi.RemoteException;

/**
 * Returns the job details for the given job ID.
 *
 * @throws InvalidJobIDException if no job by the specified job ID exists in the job scheduler
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler while
 * processing the command
 * @throws java.rmi.RemoteException
 *
 * @return the details of the job such as job ID, status text, submitter and job type
 */
public String getJobDetails(String jobid) throws
    InvalidJobIDException, SchedulerException, java.rmi.RemoteException;

/**
 * Saves the xJCL passed in to the xJCL Repository.
 *
 * @param xJCL The xJCL for the job
 * @param job The name given to the saved job in the xJCL repository. This name can
 * be used when invoking the submitJobFromRepository
 * method.
 * @param replace A boolean indicating if the xJCL in the repository should be
 * replaced, in case a job by that name already exists
 * in the xJCL repository.
 *
 * @throws InvalidOperationException if the job already exists in the xJCL
 * repository and the replace parameter specified is false
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while processing the command
 * @throws JCLException if the xJCL stored in the repository is corrupted or not valid.
 * @throws java.rmi.RemoteException
 *
 */
public void saveJobToRepository( String xJCL, String job, boolean replace ) throws
    InvalidOperationException,
    SchedulerException,
    JCLException,
    java.rmi.RemoteException;

/**
 * Returns the xJCL from the xJCL repository for the given job name.
 *
 * @param job The name given to the saved job in xJCL repository. This name can be used
 * when invoking the submitJobFromRepository
 * method.
 *
 * @throws InvalidJobNameException if job is not found in the xJCL repository.
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while processing the command
 * @throws java.rmi.RemoteException
 *
 * @return the xJCL for the given job
 */
public String showJobFromRepository( String job ) throws
    InvalidJobNameException,
    SchedulerException,
    java.rmi.RemoteException;

/**
 * Removes the xJCL for the specified job from the xJCL repository
 *
 * @param job The name given to the saved job in the xJCL repository.
 *
 * @throws InvalidJobNameException if the job is not found in the xJCL repository.
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while processing the command
 * @throws java.rmi.RemoteException

```

```

*
*/
public void removeJobFromRepository( String job ) throws
    InvalidJobNameException,
    SchedulerException,
    java.rmi.RemoteException;

/**
 * Shows all jobs in the job scheduler
 *
 * @return the list of job IDs of all jobs in the job scheduler
 *
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while processing the command
 * @throws java.rmi.RemoteException
 *
 */
public String[] showAllJobs() throws
    SchedulerException,
    java.rmi.RemoteException;

/**
 * Suspends the specified job for the number of seconds specified. Once the time period
 * is up, the job automatically
 * resumes. This only applies to batch jobs.
 *
 * @param jobid The ID of the job to suspend
 * @param seconds The number of seconds to suspend the job
 *
 * @throws InvalidJobIDException if no job by the specified job ID exists in the job scheduler
 * @throws InvalidOperationException if the operation is currently not allowed on the job
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while suspending the job
 * @throws java.rmi.RemoteException
 *
 */
public void suspendJob( String jobid, String seconds ) throws
    InvalidOperationException,
    InvalidJobIDException,
    SchedulerException,
    java.rmi.RemoteException;

/**
 * Resumes execution of the specified job. This only applies to batch jobs.
 *
 * @param jobid The ID of the job to resume
 *
 * @throws InvalidJobIDException if no job by the specified job ID exists in the
 * job scheduler
 * @throws InvalidOperationException if the operation is currently not allowed on the job
 * @throws SchedulerException if an unexpected error is thrown by the job
 * scheduler while resuming the job
 * @throws java.rmi.RemoteException
 *
 */
public void resumeJob( String jobid ) throws
    InvalidOperationException,
    InvalidJobIDException,
    SchedulerException,
    java.rmi.RemoteException;

/**
 * Returns the return code of the Batch job.
 *
 * @param jobid The ID of the job
 * @return the return code of the Batch job
 *
 * @throws InvalidJobIDException if no job by the specified job ID exists in the job scheduler
 * @throws InvalidOperationException if the operation is currently not allowed on the job
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while processing the command
 * @throws java.rmi.RemoteException

```

```

*
*/
public int getBatchJobRC(String jobid) throws InvalidOperationException,
    InvalidJobIDException, SchedulerException, java.rmi.RemoteException;

/**
 * Submits the job, which is defined by the xJCL, to the job scheduler at the specified
 * start time.
 *
 * @param xJCL The xJCL for the job
 * @param startTime The time at which the job will be submitted. The format of the
 * submit time is yyyy-mm-dd hh:mm:ss.
 * @return the job ID assigned by the job scheduler to the submitted job
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while submitting the job
 * @throws JCLException if the xJCL for the job is corrupted or not valid.
 * @throws JobSubmissionException if an error occurs while submitting the job
 * @throws InvalidStartDateTimeFormatException if the start date and/or time is
 * not in the required format
 * @throws StaleTimeException if the start date and/or time is in the past based on
 * current time
 * @throws java.rmi.RemoteException
 */
public String submitDelayedJob( String xJCL, String startTime ) throws
    SchedulerException,
    JCLException,
    JobSubmissionException,
    InvalidStartDateTimeFormatException,
    StaleTimeException,
    java.rmi.RemoteException;

/**
 * Submits the job, saved in the xJCL repository, to the job scheduler at the specified
 * start time.
 *
 * @param job The name of the job that was stored to the job repository
 * @param startTime The time at which the job will be submitted. The format of the submit
 * time is yyyy-mm-dd hh:mm:ss.
 * @return the job ID assigned by the job scheduler to the submitted job
 * @throws InvalidJobNameException if job is not found in the xJCL repository.
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while submitting the job
 * @throws JCLException if the xJCL for the job is corrupted or not valid.
 * @throws JobSubmissionException if an error occurs while submitting the job
 * @throws InvalidStartDateTimeFormatException if the start date and/or time is
 * not in the required format
 * @throws StaleTimeException if the start date and/or time is in the past based
 * on current time
 * @throws java.rmi.RemoteException
 */
public String submitDelayedJobFromRepository( String job, String startTime ) throws
    InvalidJobNameException,
    SchedulerException,
    JCLException,
    JobSubmissionException,
    InvalidStartDateTimeFormatException,
    StaleTimeException,
    java.rmi.RemoteException;

/**
 * Submits the delayed job specified by the xJCL passed in to the job scheduler and
 * saves the xJCL to the xJCL repository.
 *
 * @param xJCL The xJCL for the job
 * @param startTime The time at which the job will be submitted. The format of the
 * submit time is yyyy-mm-dd hh:mm:ss.
 * @param job The name given to the saved job in the xJCL repository. This name can
 * be used when invoking the submitJobFromRepository

```

```

* method.
* @param replace A boolean indicating if the xJCL in the repository should be replaced,
* in case a job by that name already exists
* in the job repository.
* @return the job ID assigned by the job scheduler to the submitted job
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws SchedulerException if an unexpected error is thrown by the job scheduler while
* submitting the job
* @throws JCLException if the xJCL for the job is corrupted or not valid.
* @throws JobSubmissionException if an error occurs while submitting the job
* @throws InvalidStartDateTimeFormatException if the start date and/or time is not in
* the required format
* @throws StaleTimeException if the start date and/or time is in the past based on
* current time
* @throws java.rmi.RemoteException
*/
public String saveDelayedJobToRepositoryAndSubmit( String xJCL, String job, boolean
    replace, String startTime ) throws
    InvalidOperationException,
    SchedulerException,
    JCLException,
    JobSubmissionException,
    InvalidStartDateTimeFormatException,
    StaleTimeException,
    java.rmi.RemoteException;

/**
* Creates a job schedule to submit the job, defined by the xJCL, at the specified time and
* interval.
*
* @param reqId The name of the recurring job request
* @param xJCL The xJCL for the job
* @param startTime The time at which the first job will be submitted. The format of the
* submit time is yyyy-mm-dd hh:mm:ss.
* @param interval The time interval between jobs (For example daily, weekly, monthly)
*
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws SchedulerException if an unexpected error is thrown by the job scheduler
* while submitting the job
* @throws JCLException if the xJCL for the job is corrupted or not valid.
* @throws InvalidStartDateTimeFormatException if the start date and/or time is not
* in the required format
* @throws StaleTimeException if the start date and/or time is in the past
* based on current time
* @throws InvalidIntervalException if the interval specified is not one of the
* supported time interval
* @throws java.rmi.RemoteException
*/
public void submitRecurringRequest( String reqId, String xJCL, String startTime,
    String interval ) throws
    InvalidOperationException,
    SchedulerException,
    JCLException,
    InvalidStartDateTimeFormatException,
    InvalidIntervalException,
    StaleTimeException,
    java.rmi.RemoteException;

/**
* Creates a job schedule to submit the specified job, saved in the xJCL repository, at the
* specified time and interval.
*
* @param jobName The name of the job that was stored to the job repository
* @param reqId The name of the recurring job request
* @param startTime The time at which the job will be submitted. The format of the
* submit time is yyyy-mm-dd hh:mm:ss..
* @param interval The time interval between jobs (For example daily, weekly, monthly)
*
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws SchedulerException if an unexpected error is thrown by the job scheduler
* while submitting the job

```

```

    * @throws JCLException if the xJCL for the job is corrupted or not valid.
    * @throws InvalidStartDateTimeFormatException if the start date and/or time is not
    * in the required format
    * @throws StaleTimeException if the start date and/or time is in the
    * past based on current time
    * @throws InvalidIntervalException if the interval specified is not one of the supported
    * time interval
    * @throws InvalidJobNameException if job is not found in the xJCL repository.
    * @throws java.rmi.RemoteException
    */
public void submitRecurringRequestFromRepository
    (String jobName, String reqId, String startTime,
    String interval) throws
    InvalidOperationException,
    SchedulerException,
    JCLException,
    InvalidStartDateTimeFormatException,
    InvalidIntervalException,
    StaleTimeException,
    InvalidJobNameException,
    java.rmi.RemoteException;

/**
 * Cancel an existing job schedule
 *
 * @param reqId The name of the job schedule
 * @throws InvalidOperationException if the operation is currently not allowed on the job
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler while
 * canceling the job
 * @throws java.rmi.RemoteException
 */
public void cancelRecurringRequest( String reqId ) throws
    InvalidOperationException,
    SchedulerException,
    java.rmi.RemoteException;

/**
 * Returns details of an existing job schedule.
 *
 * @param reqId The name of the job schedule to be returned
 * @return information about the schedule such as schedule name, job name, start time and
 * interval
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler while
 * processing the command
 * @throws InvalidOperationException if the operation is currently not allowed on the job
 * @throws java.rmi.RemoteException
 */
public String getRecurringRequestDetails(String reqId) throws
    SchedulerException,
    InvalidOperationException,
    java.rmi.RemoteException;

/**
 * Modify an existing job schedule.
 *
 * @param reqId The name of the job schedule to be modified
 * @param xJCL The xJCL for the job
 * @param startTime The time at which the first job will be submitted. The format of the
 * submit time is yyyy-mm-dd hh:mm:ss.
 * @param interval The time interval between jobs (For example daily, weekly, monthly)
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while processing the command
 * @throws JCLException if the xJCL for the job is corrupted or not valid.
 * @throws InvalidOperationException if the operation is currently not allowed on the job
 * @throws InvalidStartDateTimeFormatException if the start date and/or time is not in the
 * required format
 * @throws StaleTimeException if the start date and/or time is in the past based on
 * current time
 * @throws InvalidIntervalException if the interval specified is not one of the supported
 * time interval

```

```

    * @throws java.rmi.RemoteException
    */
public void modifyRecurringRequest(String reqId, String xJCL, String startTime,
    String interval) throws
    SchedulerException,
    JCLEException,
    InvalidOperationException,
    InvalidStartDateTimeFormatException,
    StaleTimeException,
    InvalidIntervalException,
    java.rmi.RemoteException;

/**
 * Lists all existing job schedules
 *
 * @return a list of all job schedules currently in the system
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler while
 * processing the command
 * @throws java.rmi.RemoteException
 */
public String[] showAllRecurringRequests() throws
    SchedulerException,
    java.rmi.RemoteException;

/**
 * Show all jobs in the specified job schedule
 *
 * @param reqId the name of the job schedule
 * @return the list of job IDs of jobs in the specified job schedule
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while processing the command
 * @throws InvalidOperationException if the operation is currently not allowed on the job
 * @throws java.rmi.RemoteException
 */
public String[] showRecurringJobs(String reqId) throws
    SchedulerException,
    InvalidOperationException,
    java.rmi.RemoteException;

/**
 * Returns job status in XML format for the given job IDs.
 *
 * @param jobid List of job IDs
 *
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while processing the command
 * @throws java.rmi.RemoteException
 *
 * @return Job status such as job ID, return code, status code and status text in XML format
 */
public String getJobsStatus(String[] jobid) throws
    SchedulerException,
    java.rmi.RemoteException;

/**
 * Returns a list of job IDs that match the specified criteria. All conditions must apply
 * for a match to occur.
 *
 * @param jobFilter SQL filter value to apply to the job ID (For example Postings% )
 * @param submitterFilter SQL filter value to apply to the submitter
 * @param nodeFilter SQL filter value to apply to the names of the nodes where the
 * jobs executed (For example node_ )
 * @param appServerFilter SQL filter value to apply to the names of the application
 * servers where the jobs executed
 * @param stateFilter List of job states. Refer to {@link JobStatusConstants
 * JobStatusConstants} for a
 * list of the possible job states.
 * @param sortBy - Field used to sort results (For example JOBID, STATUS, APPSERVER)
 * @param ascending - flag indicating whether the results should be returned in
 * ascending or descending order
 * of the sortBy field.

```

```

*
* @return the list of job IDs that match the specified criteria
*
* @throws SchedulerException if an unexpected error is thrown by the job scheduler
* while processing the command
* @throws java.rmi.RemoteException
*/
public String[] getJobsId(String jobFilter, String submitterFilter,
    String nodeFilter, String appServerFilter, Integer[] stateFilter, String sortBy,
    boolean ascending) throws
SchedulerException,
java.rmi.RemoteException;

/**
 * Cancels the jobs identified by the list of job IDs
 *
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while canceling the job
 * @throws java.rmi.RemoteException
 *
 * @param jobid The list of job IDs to cancel
 * @return List of return codes. Refer to {@link JobSchedulerConstants JobSchedulerConstants}
 * for a list of the possible return codes.
 */
public int[] cancelJob( String[] jobid ) throws
SchedulerException,
java.rmi.RemoteException;

/**
 * Purges the jobs, identified by the list of job IDs, from the job scheduler and the
 * grid endpoint environments.
 *
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while purging the job
 * @throws java.rmi.RemoteException
 *
 * @param jobid The list of job IDs to purge
 * @return List of return codes. Refer to
 * {@link JobSchedulerConstants JobSchedulerConstants} for a list of the possible
 * return codes.
 */
public int[] purgeJob( String[] jobid ) throws
SchedulerException,
java.rmi.RemoteException;

/**
 * Restarts the jobs identified by the list of job IDs. Only jobs in the
 * restartable state can be restarted.
 *
 * @throws SchedulerException if an unexpected error is thrown by the
 * job scheduler while restarting the job
 * @throws java.rmi.RemoteException
 *
 * @param jobid The list of job IDs to restart
 * @return List of return codes. Refer to {@link JobSchedulerConstants
 * JobSchedulerConstants} for a list of the possible return codes.
 */
public int[] restartJob( String[] jobid ) throws
SchedulerException,
java.rmi.RemoteException;

/**
 * Resumes execution of the jobs identified by the list of job IDs. This only
 * applies to batch jobs.
 *
 * @param jobid The list of job IDs to resume
 * @return List of return codes. Refer to {@link JobSchedulerConstants
 * JobSchedulerConstants} for a list of the possible return codes.
 *
 * @throws SchedulerException if an unexpected error is thrown by the job
 * scheduler while resuming the job
 * @throws java.rmi.RemoteException
 *

```



```

*/
public int[] resumeJob( String[] jobid ) throws
SchedulerException,
java.rmi.RemoteException;

/**
 * Suspends the specified jobs for the number of seconds specified. Once the
 * time period is up, the jobs automatically
 * resume. This only applies to batch jobs.
 *
 * @param jobid The ID of the job to suspend
 * @param seconds The number of seconds to suspend the job
 * @return List of return codes. Refer to {@link JobSchedulerConstants
 * JobSchedulerConstants} for a list of the possible return codes.
 *
 * @throws InvalidOperationException if the operation is currently not allowed on the job
 * @throws SchedulerException if an unexpected error is thrown by the
 * job scheduler while suspending the job
 * @throws java.rmi.RemoteException
 *
 */
public int[] suspendJob( String[] jobid, String seconds ) throws
SchedulerException,
InvalidOperationException,
java.rmi.RemoteException;

/**
 *
 * Submits the specified job, saved in the xJCL repository, and any name/value pairs
 * specified to the job scheduler at the specified
 * start time.
 *
 * @param job The name of the job that was stored to the xJCL repository
 * @param startTime The time at which the job will be submitted. The format of the submit
 * time is yyyy-mm-dd hh:mm:ss.
 * @param nameValuePairs The space delimited name=value pairs which are used to
 * modify the xJCL For example. "host=myhost port=myport")
 * Any values that contain special characters or spaces must be URL encoded with an
 * encoding scheme of UTF-8 before being passed in on the request.
 *
 * @return the job ID assigned by the job scheduler to the submitted job
 *
 * @throws InvalidJobNameException if the job is not found in the xJCL repository.
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while submitting the job
 * @throws JCLException if the xJCL for the job is corrupted or not valid.
 * @throws JobSubmissionException if an error occurs while submitting the job
 * @throws InvalidStartDateTimeFormatException if the start date and/or time is not
 * in the required format
 * @throws StaleTimeException if the start date and/or time is in the past
 * based on current time
 * @throws java.rmi.RemoteException
 */
public String submitModifiableDelayedJobFromRepository( String job, String startTime,
String nameValuePairs )
throws InvalidJobNameException, SchedulerException, JCLException, JobSubmissionException,
InvalidStartDateTimeFormatException, StaleTimeException, java.rmi.RemoteException;

/**
 *
 * Submits the job, which is defined by the xJCL and any name/value pairs specified, to
 * the job scheduler at the specified
 * start time.
 *
 * @param xJCL The xJCL for the job
 * @param startTime The time at which the job will be submitted. The format of the
 * submit time is yyyy-mm-dd hh:mm:ss.
 * @param nameValuePairs The space delimited name=value pairs which are used to
 * modify the xJCL For example. "host=myhost port=myport")
 * Any values that contain special characters or spaces must be URL encoded with
 * an encoding scheme of UTF-8 before being passed in on the request.
 *
 * @return the job ID assigned by the job scheduler to the submitted job

```

```

*
* @throws SchedulerException if an unexpected error is thrown by the job
* scheduler while submitting the job
* @throws JCLException if the xJCL for the job is corrupted or not valid.
* @throws JobSubmissionException if an error occurs while submitting the job
* @throws InvalidStartDateTimeFormatException if the start date and/or time is
* not in the required format
* @throws StaleTimeException if the start date and/or time is in the past
* based on current time
* @throws java.rmi.RemoteException
*/

public String submitModifiableDelayedJob
( String xJCL, String startTime, String nameValuePairs )
throws SchedulerException, JCLException, JobSubmissionException,
    InvalidStartDateTimeFormatException, StaleTimeException, java.rmi.RemoteException;

/**
 * Submits the delayed job, which is defined by the xJCL and any name/value pairs
 * specified, to the job scheduler and
 * saves the xJCL to the xJCL repository.
 *
 * @param xJCL The xJCL for the job
 * @param startTime The time at which the job will be submitted. The format of the
 * submit time is yyyy-mm-dd hh:mm:ss.
 * @param job The name given to the saved job in the xJCL repository. This name
 * can be used when invoking the submitJobFromRepository
 * method.
 * @param replace A boolean indicating if the xJCL in the repository should be
 * replaced, in case a job by that name already exists
 * in the job repository.
 * @param nameValuePairs The space delimited name=value pairs which are used
 * to modify the xJCL For example. "host=myhost port=myport")
 * Any values that contain special characters or spaces must be URL encoded
 * with an encoding scheme of UTF-8 before being passed in on the request.
 *
 * @return the job ID assigned by the job scheduler to the submitted job
 *
 * @throws InvalidOperationException if the operation is currently not allowed on the job
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while submitting the job
 * @throws JCLException if the xJCL for the job is corrupted or not valid.
 * @throws JobSubmissionException if an error occurs while submitting the job
 * @throws InvalidStartDateTimeFormatException if the start date and/or time is
 * not in the required format
 * @throws StaleTimeException if the start date and/or time is in the past based
 * on current time
 * @throws java.rmi.RemoteException
 */

public String saveModifiableDelayedJobToRepositoryAndSubmit
( String xJCL, String job, boolean replace, String startTime, String nameValuePairs )
throws InvalidOperationException, SchedulerException, JCLException, JobSubmissionException,
    InvalidStartDateTimeFormatException, StaleTimeException, java.rmi.RemoteException;

/**
 * Creates a job schedule to submit jobs at the specified time interval. The jobs
 * are defined by the xJCL and any name/value pairs specified.
 *
 * @param reqId The name of the job schedule
 * @param xJCL The xJCL for the job
 * @param startTime The time at which the job will be submitted. The format of
 * the submit time is yyyy-mm-dd hh:mm:ss.
 * @param interval The time interval between jobs (For example daily, weekly, monthly)
 * @param nameValuePairs The space delimited name=value pairs which are used to
 * modify the xJCL For example. "host=myhost port=myport")
 * Any values that contain special characters or spaces must be URL encoded
 * with an encoding scheme of UTF-8 before being passed in on the request.
 *
 * @throws InvalidOperationException if the operation is currently not
 * allowed on the job
 * @throws SchedulerException if an unexpected error is thrown by the

```

```

* job scheduler while submitting the job
* @throws JCLEException if the xJCL for the job is corrupted or not valid.
* @throws InvalidStartDateTimeFormatException if the start date and/or time
* is not in the required format
* @throws StaleTimeException if the start date and/or time is in the past
* based on current time
* @throws InvalidIntervalException if the interval specified is not one of
* the supported time interval
* @throws java.rmi.RemoteException
*/

public void submitModifiableRecurringRequest
( String reqId, String xJCL, String startTime, String interval, String nameValuePairs )
throws InvalidOperationException, SchedulerException, JCLEException,
InvalidStartDateTimeFormatException,
    InvalidIntervalException, StaleTimeException, java.rmi.RemoteException;

/**
* Creates a job schedule to submit jobs at the specified time interval. The jobs
* are defined by the xJCL stored in the xJCL repository
* and any name/value pairs specified.
*
* @param jobName The name of the job that was stored to the job repository
* @param reqId The name of the recurring job request
* @param startTime The time at which the job will be submitted. The format of
* the submit time is yyyy-mm-dd hh:mm:ss.
* @param interval The time interval between jobs (For example daily, weekly, monthly)
* @param nameValuePairs The space delimited name=value pairs which are used to
* modify the xJCL For example. "host=myhost port=myport")
* Any values that contain special characters or spaces must be URL encoded with
* an encoding scheme of UTF-8 before being passed in on the request.
*
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws SchedulerException if an unexpected error is thrown by the job scheduler
* while submitting the job
* @throws JCLEException if the xJCL for the job is corrupted or not valid.
* @throws InvalidStartDateTimeFormatException if the start date and/or time is not
* in the required format
* @throws StaleTimeException if the start date and/or time is in the past
* based on current time
* @throws InvalidIntervalException if the interval specified is not one of the
* supported time interval
* @throws InvalidJobNameException if job is not found in the xJCL repository.
* @throws java.rmi.RemoteException
*/

public void submitModifiableRecurringRequestFromRepository(String jobName, String
reqId, String startTime, String interval, String nameValuePairs )
throws InvalidOperationException, SchedulerException, JCLEException,
InvalidStartDateTimeFormatException,
    InvalidIntervalException, StaleTimeException, InvalidJobNameException,
    java.rmi.RemoteException;

/**
* Submits the job, which is defined by the xJCL and any name/value pairs
* specified, to the job scheduler and
* saves the xJCL to the xJCL repository.
*
* @param xJCL The xJCL for the job
* @param job The name given to the saved job in xJCL repository. This name
* can be used when invoking the submitJobFromRepository
* method.
* @param replace A boolean indicating if the xJCL in the repository
* should be replaced, in case a job by that name already exists
* in the xJCL repository.
* @param nameValuePairs The space delimited name=value pairs which are
* used to modify the xJCL For example. "host=myhost port=myport")
* Any values that contain special characters or spaces must be URL encoded
* with an encoding scheme of UTF-8 before being passed in on the request.
*
* @return the job ID assigned by the job scheduler to the submitted job
*
*/

```

```

* @throws InvalidOperationException if the job already exists in the xJCL
* repository and the replace parameter specified is false
* @throws SchedulerException if an unexpected error is thrown by the
* job scheduler while submitting the job
* @throws JCLException if the xJCL stored in the repository is corrupted or not valid.
* @throws JobSubmissionException if an error occurs while submitting the job
* @throws java.rmi.RemoteException
*
*/

public String saveModifiableJobToRepositoryAndSubmit( String xJCL, String job,
    boolean replace, String nameValuePairs )
throws InvalidOperationException, SchedulerException, JCLException,
    JobSubmissionException, java.rmi.RemoteException;

/**
* Submits the specified job, saved in the xJCL repository, and any name/value
* pairs specified to the job scheduler
*
* @param job The name of the job that was stored to the xJCL repository
* @param nameValuePairs The space delimited name=value pairs which are used to
* modify the xJCL (For example. "host=myhost port=myport")
* Any values that contain special characters or spaces must be URL encoded with
* an encoding scheme of UTF-8 before being passed in on the request.
* @return the job ID assigned by the job scheduler to the submitted job
*
* @throws InvalidJobNameException if job is not found in the xJCL repository.
* @throws SchedulerException if an unexpected error is thrown by the job
* scheduler while submitting the job
* @throws JCLException if the xJCL stored in the repository is corrupted or not valid.
* @throws JobSubmissionException if an error occurs while submitting the job
* @throws java.rmi.RemoteException
*/

public String submitModifiableJobFromRepository( String job, String nameValuePairs )
throws InvalidJobNameException, SchedulerException, JCLException, JobSubmissionException,
    java.rmi.RemoteException;

/**
* Submits the job, which is defined by the xJCL and any name/value pairs specified,
* to the job scheduler
*
* @param xJCL The xJCL for the job
* @param nameValuePairs The space delimited name=value pairs which are used to
* modify the xJCL (For example "host=myhost port=myport")
* Any values that contain special characters or spaces must be URL encoded
* with an encoding scheme of UTF-8 before being passed in on the request.
* @return the job ID assigned by the job scheduler to the submitted job
*
* @throws SchedulerException if an unexpected error is thrown by the job
* scheduler while submitting the job
* @throws JCLException if the xJCL stored in the repository is corrupted or not valid.
* @throws JobSubmissionException if an error occurs while submitting the job
* @throws java.rmi.RemoteException
*
*/

public String submitModifiableJob( String xJCL, String nameValuePairs )
throws SchedulerException, JCLException, JobSubmissionException, java.rmi.RemoteException;

/**
* Modify an existing job schedule.
*
* @param reqId The name of the job schedule to be modified
* @param xJCL The xJCL for the job
* @param startTime The time at which the first job
* will be submitted. The format of the submit time is yyyy-mm-dd hh:mm:ss.
* @param interval The time interval between jobs
* (For example daily, weekly, monthly)
* @param nameValuePairs The space delimited name=value
* pairs which are used to modify the xJCL (For example "host=myhost port=myport")
* Any values that contain special characters or spaces must be URL encoded with

```

```

*   an encoding scheme of UTF-8 before being passed in on the request.
* @throws SchedulerException         if an unexpected error is thrown
*   by the job scheduler while processing the command
* @throws JCLException               if the xJCL for the job is corrupted
*   or not valid.
* @throws InvalidOperationException if the operation is currently not
*   allowed on the job
* @throws InvalidStartDateTimeFormatException if the start date and/or time is
*   not in the required format
* @throws StaleTimeException         if the start date and/or time is
*   in the past based on current time
* @throws InvalidIntervalException  if the interval specified is not
*   one of the supported time interval
* @throws java.rmi.RemoteException
*/

public void modifyModifiableRecurringRequest(String reqId, String xJCL,
      String startTime, String interval, String nameValuePairs)
throws SchedulerException, JCLException, InvalidOperationException,
      InvalidStartDateTimeFormatException, StaleTimeException, InvalidIntervalException,
      java.rmi.RemoteException;

/**
 * Returns a list of job names in the job repository that match the specified
 * criteria. All conditions must apply for a match to occur.
 *
 * @param jobNameFilter SQL filter value to apply to the job names (For example Postings% )
 * @param jobDescFilter not used
 * @param sortBy - Field used to sort results (For example JOBNAME, TXT)
 * @param ascending - flag indicating whether the results should be returned in
 * ascending or descending order
 * of the sortBy field.
 *
 * @return the list of job names that match the specified criteria
 *
 * @throws SchedulerException if an unexpected error is thrown by the job
 * scheduler while processing the request
 * @throws java.rmi.RemoteException
 */
public String[] getJobsName(String jobNameFilter, String jobDescFilter,
      String sortBy, boolean ascending) throws
SchedulerException,
java.rmi.RemoteException;

/**
 * Stops the job identified by the job ID
 *
 * @throws InvalidOperationException if the operation is currently not
 * allowed on the job
 * @throws InvalidJobIDException if no job by the specified job ID exists
 * in the job scheduler
 * @throws SchedulerException if an unexpected error is thrown by the job
 * scheduler while processing the request
 * @throws java.rmi.RemoteException
 *
 * @param jobid The ID of the job
 */
public void stopJob( String jobid )
throws InvalidOperationException, InvalidJobIDException, SchedulerException,
      java.rmi.RemoteException;

/**
 * Stops the jobs identified by the list of job IDs
 *
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while purging the job
 * @throws java.rmi.RemoteException
 *
 * @param jobid The list of job IDs to stop
 * @return List of return codes. Refer to {@link JobSchedulerConstants
 * JobSchedulerConstants} for a list of the possible return codes.
 */

```

```

public int[] stopJob( String[] jobid ) throws
SchedulerException,
java.rmi.RemoteException;

/**
 * Parses the xJCL to produce a map of all symbolic variables used in the xJCL
 * which are not system properties
 * @param xJCL The xJCL for the job
 * @return a map of defaulted name/value pairs; value==null ==> no default value
 * in substitution-props
 *
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while processing the command
 * @throws JCLException if the xJCL stored in the repository is corrupted or not valid.
 * @throws java.rmi.RemoteException
 *
 */

public String getSymbolicVariables( String clientXJCL )
throws SchedulerException, JCLException, java.rmi.RemoteException;

/**
 * Returns job schedule information in XML format for the given job schedule names.
 *
 * @param requestid List of job schedule names
 *
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while processing the command
 * @throws java.rmi.RemoteException
 *
 * @return Job schedule information in XML format, such as job schedule name,
 * job name, start time and interval
 */
public String getRequests(String[] requestid) throws
SchedulerException,
java.rmi.RemoteException;

/**
 * Returns a list of job schedule names that match the specified criteria. All conditions
 * must apply for a match to occur.
 *
 * @param requestIdFilter SQL filter value to apply to the name of the job schedule
 * (For example %Postings%)
 * @param startTimeFilter SQL filter value to apply to the initial submit time of the
 * jobs. The format of the submit time is yyyy-mm-dd hh:mm:ss.
 * @param submitterFilter SQL filter value to apply to the submitter
 * @param intervalFilter List of time periods between job submissions (For example daily,
 * weekly, monthly)
 * @param statusFilter List of job states. Refer to
 * {@link JobStatusConstants JobStatusConstants} for a
 * list of the possible job states.
 * @param sortBy - Field used to sort results (For example REQUESTID, STARTTIME, INTERVAL)
 * @param ascending - flag indicating whether the results should be returned in
 * ascending or descending order
 * of the sortBy field.
 *
 * @return the list of job schedule names that match the specified criteria
 *
 * @throws SchedulerException if an unexpected error is thrown by the job scheduler
 * while processing the command
 * @throws java.rmi.RemoteException
 */
public String[] getRequestsId(String requestIdFilter, String startTimeFilter,
String submitterFilter, String[] intervalFilter, Integer[] statusFilter,
String sortBy, boolean ascending) throws
SchedulerException,
java.rmi.RemoteException;

/**
 * Cancel existing job schedules
 *
 * @param reqId The list of job schedule names to cancel

```

```

* @return List of return codes. Refer to
* {@link JobSchedulerConstants JobSchedulerConstants} for a list of the
* possible return codes.
*
* @throws SchedulerException if an unexpected error is thrown by the job scheduler
* while canceling the job
* @throws java.rmi.RemoteException
*/
public int[] cancelRecurringRequest( String[] reqId ) throws
SchedulerException,
java.rmi.RemoteException;

/**
* Returns the compressed job log associated with the requested job ID
*
* @param jobId The ID of the job whose log file name is to be returned
* @return the file system name for the job log of the specified job
* @throws SchedulerException if an unexpected error is thrown by the Job Scheduler while
* processing the command
* @throws InvalidJobIDException if no job logs for the specified job ID are found by
* the Job Scheduler
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws java.rmi.RemoteException
*/
public String getJobLog( String jobId ) throws SchedulerException,
InvalidJobIDException, InvalidOperationException, java.rmi.RemoteException;

/**
* Returns the job log meta-data associated with the requested job ID
* list of distinct job log subdirectories for the job ID)
*
* @param jobId The ID of the job whose meta-data is to be returned
* @return the job log meta-data for the specified job
* @throws SchedulerException if an unexpected error is thrown by the Job Scheduler
* while processing the command
* @throws InvalidJobIDException if no job log meta-data for the specified job ID is
* found by the Job Scheduler
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws java.rmi.RemoteException
*/
public String[] getLogMetaData( String jobId ) throws SchedulerException,
InvalidJobIDException, InvalidOperationException, java.rmi.RemoteException;

/**
* Returns the job log part list associated with the requested job ID and log subdirectory
*
* @param jobId The ID of the job whose part information is to be returned
* @param logSubDirName The name of the log subdirectory of the job whose part
* information is to be returned
* @return the job log part information for the specified job
* @throws SchedulerException if an unexpected error is thrown by the Job Scheduler
* while processing the command
* @throws InvalidJobIDException if no part information for the specified job ID is
* found by the Job Scheduler
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws java.rmi.RemoteException
*/
public String[] getLogPartList( String jobId, String logSubDirName ) throws
SchedulerException, InvalidJobIDException, InvalidOperationException,
java.rmi.RemoteException;

/**
* Returns the contents of the job log file associated with the requested job ID,
* log subdirectory and part number
*
* @param jobId The ID of the job whose part information is to be returned
* @param logSubDirName The name of the log subdirectory of the job whose part
* information is to be returned
* @param partNumber The name of the job log chunk in the log subdirectory whose
* part information is to be returned
*
* @return the contents of the job log part for the specified job and log subdirectory

```



```

*
* @throws SchedulerException if an unexpected error is thrown by the Job Scheduler
* while processing the command
* @throws InvalidJobIDException if no part information for the specified job ID is
* found by the Job Scheduler
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws java.rmi.RemoteException
*/
public String[] getLogPart( String jobid, String logSubDirName, String partNumber )
    throws SchedulerException, InvalidJobIDException, InvalidOperationException,
        java.rmi.RemoteException;

/**
* Returns the size in bytes of the job log file associated with the requested job ID
*
* @param jobid The ID of the job whose size information is to be returned
* @param logSubDirName The name of the log subdirectory of the job whose size
* information is to be returned
* @return the size of the job log for the specified job
* @throws SchedulerException if an unexpected error is thrown by the Job Scheduler
* while processing the command
* @throws InvalidJobIDException if no size information for the specified job ID is
* found by the Job Scheduler
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws java.rmi.RemoteException
*/
public String getLogSize( String jobid, String logSubDirName ) throws SchedulerException,
    InvalidJobIDException, InvalidOperationException, java.rmi.RemoteException;

/**
* Returns the age of the job log file associated with the requested job ID and log
* subdirectory
*
* @param jobid The ID of the job whose age information is to be returned
* @param logSubDirName The name of the log subdirectory of the job whose age information
* is to be returned
*
* @return the age of the job log in days for the specified jobname and log subdirectory
*
* @throws SchedulerException if an unexpected error is thrown by the Job Scheduler
* while processing the command
* @throws InvalidJobIDException if no age information for the specified job ID is
* found by the Job Scheduler
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws java.rmi.RemoteException
*/
public int getLogAge( String jobid, String logSubDirName ) throws SchedulerException,
    InvalidJobIDException, InvalidOperationException, java.rmi.RemoteException;

/**
* Returns the job log list associated with the requested job class
*
* @param jobid The class identifier whose job list information is to be returned
* @param jobClass The class identifier on which to match
* @return a list of all job IDs whose class identifier matches the specified jobClass
* @throws SchedulerException if an unexpected error is thrown by the Job Scheduler
* while processing the command
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws java.rmi.RemoteException
*/
public String[] getJobsByClass( String jobClass ) throws SchedulerException,
    InvalidOperationException, java.rmi.RemoteException;

/**
* Removes the compressed job log associated with the requested job ID [ this is the
* required complimentary action to {@link JobScheduler#getJobLog(String) getJobLog(jobid) }
*
* @param jobid The ID of the job whose compressed log file is to be removed
* @throws SchedulerException if an unexpected error is thrown by the Job Scheduler
* while processing the command
* @throws InvalidJobIDException if no part information for the specified job ID is

```



```

* found by the Job Scheduler
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws java.rmi.RemoteException
*/
public void removeJobLog( String jobId ) throws SchedulerException,
    InvalidJobIDException, InvalidOperationException, java.rmi.RemoteException;

/**
* Purges the job log file associated with the requested job ID and log subDirectory
*
* @param jobId The ID of the job whose job log is to be purged
* @param logSubDirName The name of the log subdirectory of the job whose job log
* is to be purged
* @throws SchedulerException if an unexpected error is thrown by the Job Scheduler
* while processing the command
* @throws InvalidJobIDException if no job information for the specified job ID is
* found by the Job Scheduler
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws java.rmi.RemoteException
*/
public void purgeJobLog( String jobId, String logSubDirName ) throws SchedulerException,
    InvalidJobIDException, InvalidOperationException, java.rmi.RemoteException;

/**
* Return the JMX addresses of the scheduler cluster
*
* @return the JMX addresses of the scheduler cluster
* @throws SchedulerException if an unexpected error is thrown by the Job Scheduler
* while processing the command
* @throws InvalidOperationException if the operation is currently not allowed on the job
* @throws java.rmi.RemoteException
*/
public String[] getAdminAddresses() throws SchedulerException, InvalidOperationException,
    java.rmi.RemoteException;

/**
* Retrieves a list of user preferences for the given user ID and the given scope.
* @param userId The user ID used to log into the Job Management Console
* @param prefScope The scope of the preferences within the Job Management Console.
* (For example JobCollectionForm, SavedJobCollectionForm,
* JobScheduleCollectionForm)
* @return a list of user preferences in the format of name=value
*
* @throws SchedulerException if an unexpected error is thrown by the job scheduler
* while processing the command
* @throws java.rmi.RemoteException
*/
public String[] getUserPrefs( String userId, String prefScope ) throws
    SchedulerException, java.rmi.RemoteException;

/**
* Saves the list of user preferences for the given user ID and the given scope.
* @param userId The user ID used to log into the Job Management Console
* @param prefScope The scope of the preferences within the Job Management Console.
* (For example JobCollectionForm, SavedJobCollectionForm,
* JobScheduleCollectionForm)
* @param prefNameValue The list of user preferences in the format of name=value
*
* @throws SchedulerException if an unexpected error is thrown by the job scheduler
* while processing the command
* @throws java.rmi.RemoteException
*/
public void saveUserPrefs( String userId, String prefScope, String[] prefNameValue)
    throws SchedulerException, java.rmi.RemoteException;

/**
* Returns the job log list associated with the requested job class sorted by job log age
*
* @param jobClass The class identifier on which to match
* @return a list of all job IDs whose class identifier matches jobClass

```

```

    * @throws SchedulerException if an unexpected error is thrown by the Job Scheduler
    * while processing the command
    * @throws InvalidOperationException if the operation is currently not allowed on the job
    * @throws java.rmi.RemoteException
    */
public String[] getJobLogMetaDataByAgeForClass( String jobClass ) throws
    SchedulerException, InvalidOperationException, java.rmi.RemoteException;

/**
 * Returns the job log list associated with the requested job class sorted by job log size
 *
 * @param jobClass The class identifier on which to match
 * @return a list of all job IDs whose class identifier matches jobClass
 * @throws SchedulerException if an unexpected error is thrown by the Job Scheduler
 * while processing the command
 * @throws InvalidOperationException if the operation is currently not allowed on the job
 * @throws java.rmi.RemoteException
 */
public String[] getJobLogMetaDataBySizeForClass( String jobClass ) throws
    SchedulerException, InvalidOperationException, java.rmi.RemoteException;

/**
 * Stops user job logging
 *
 * @param jobId The ID of the job whose application job logging is to be stopped
 * @throws SchedulerException if an unexpected error is thrown by the Job Scheduler
 * while processing the command
 * @throws InvalidJobIDException if the specified job ID is not found by the Job Scheduler
 * @throws InvalidOperationException if the operation is currently not allowed on the job
 * @throws java.rmi.RemoteException
 */
public void quiesceLogging( String jobId ) throws SchedulerException,
    InvalidJobIDException, InvalidOperationException, java.rmi.RemoteException;

/**
 *
 *
 * @param jobId The ID of the job
 * @param Status The status of the job
 *
 * @throws java.rmi.RemoteException
 */
public void sendCheckpointNotification( String jobId, String Status )
    throws java.rmi.RemoteException;

/**
 * Returns true if SAF authorization is enabled.
 * Supported on z/OS only.
 *
 * @return true if SAF authorization is enabled, otherwise false.
 * @throws java.rmi.RemoteException
 */
public boolean isSAF() throws java.rmi.RemoteException;
}

```

Submitting batch jobs using the job scheduler web service interface

The job scheduler web service interface is used to programmatically submit and manipulate a batch job.

Before you begin

The job scheduler supports programmatic access to its functions over both an EJB interface for Java Platform, Enterprise Edition (Java EE) applications and a web service interface for both Java EE and non-Java EE applications. The Web Services Description Language (WSDL) describes the web service interface for the job scheduler.

Develop and install your batch applications.

About this task

This topic describes how to submit a batch job to the job scheduler. It includes a code example that demonstrates how to invoke the job scheduler web service interface.

Procedure

1. Create a program for submitting batch work.

The following example demonstrates how to invoke the job scheduler web service interface to submit a batch job.

Some statements are split on multiple lines for printing purposes.

```
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceException;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.encoding.XMLType;

    Call call = null;
    String lrsHostName = "localhost";
    String lrsPort = "9080";

private String readXJCL() throws FileNotFoundException, IOException {
    // Code to read xJCL file into a String
}

    public void submitJob() {
        String endPoint =
            "http://"+lrsHostName+": "+lrsPort+"/LongRunningJobSchedulerWebSvcRouter/
            services/JobScheduler";
        try {
            ServiceFactory serviceFactory = ServiceFactory.newInstance();
            Service service = serviceFactory.createService(new
                QName("http://longrun.websphere.ibm.com", "JobSchedulerService"));

            call = (Call) service.createCall();
            call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,
                "http://schemas.xmlsoap.org/soap/encoding/");
            call.setProperty(Call.OPERATION_STYLE_PROPERTY, "wrapped");
            call.setPortTypeName(new
                QName("http://longrun.websphere.ibm.com", "JobSchedulerService"));
            call.setTargetEndpointAddress(endPoint);

            //remove all parameters from call object
            call.removeAllParameters();
            call.setReturnType(XMLType.SOAP_STRING, null);
            call.addParameter("arg", XMLType.SOAP_STRING, ParameterMode.IN);
            call.setOperationName(new QName("http://longrun.websphere.ibm.com", "submitJob"));

String xjcl = readXJCL(); // Method to read xJCL file into a string

            call.invoke(new Object[] {xjcl});
        } catch (ServiceException se) {
            System.out.println("Service Exception: " + se.getMessage());
            se.printStackTrace();
        } catch (java.rmi.RemoteException re) {
            System.out.println("Remote Exception: " + re.getMessage());
            re.printStackTrace();
        }
    }
}
```

2. Run the program to submit batch work.

Job scheduler web service interface

The job scheduler for web services provides the following interfaces for programmatically submitting and manipulating a batch job from a web services client program:

The following code is the contents of the Web Services Description Language (WSDL) file.

Some code is split on multiple lines for printing purposes.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://longrun.websphere.ibm.com"
xmlns:impl="http://longrun.websphere.ibm.com"
xmlns:intf="http://longrun.websphere.ibm.com"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:ws-i="http://ws-i.org/profiles/basic/1.1/xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
    <schema
targetNamespace="http://longrun.websphere.ibm.com"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:impl="http://longrun.websphere.ibm.com"
xmlns:intf="http://longrun.websphere.ibm.com"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <element name="submitJobFromRepositoryResponse">
        <complexType>
          <sequence>
            <element name="submitJobFromRepositoryReturn" nillable="true" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
      <element name="submitJob">
        <complexType>
          <sequence>
            <element name="arg_0_1" nillable="true" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
      <element name="submitJobResponse">
        <complexType>
          <sequence>
            <element name="submitJobReturn" nillable="true" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
      <element name="showAllJobs">
        <complexType>
          <sequence/>
        </complexType>
      </element>
      <element name="showAllJobsResponse">
        <complexType>
          <sequence>
            <element name="showAllJobsReturn" nillable="true" type="impl:ArrayOf_xsd_nillable_string"/>
          </sequence>
        </complexType>
      </element>
      <element name="saveJobToRepository">
        <complexType>
          <sequence>
            <element name="arg_0_3" nillable="true" type="xsd:string"/>
            <element name="arg_1_3" nillable="true" type="xsd:string"/>
            <element name="arg_2_3" type="xsd:boolean"/>
          </sequence>
        </complexType>
      </element>
      <element name="saveJobToRepositoryResponse">
        <complexType>
          <sequence/>
        </complexType>
      </element>
      <element name="suspendJob">
        <complexType>
          <sequence>
            <element name="arg_0_4" nillable="true" type="xsd:string"/>
            <element name="arg_1_4" nillable="true" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
    </schema>
  </wsdl:types>
</wsdl:definitions>
```

```

    </sequence>
  </complexType>
</element>
<element name="suspendJobResponse">
  <complexType>
    <sequence/>
  </complexType>
</element>
<element name="modifyRecurringRequest">
  <complexType>
    <sequence>
      <element name="arg_0_5" nillable="true" type="xsd:string"/>
      <element name="arg_1_5" nillable="true" type="xsd:string"/>
      <element name="arg_2_5" nillable="true" type="xsd:string"/>
      <element name="arg_3_5" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="modifyRecurringRequestResponse">
  <complexType>
    <sequence/>
  </complexType>
</element>
<element name="saveDelayedJobToRepositoryAndSubmit">
  <complexType>
    <sequence>
      <element name="arg_0_6" nillable="true" type="xsd:string"/>
      <element name="arg_1_6" nillable="true" type="xsd:string"/>
      <element name="arg_2_6" type="xsd:boolean"/>
      <element name="arg_3_6" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="saveDelayedJobToRepositoryAndSubmitResponse">
  <complexType>
    <sequence>
      <element name="saveDelayedJobToRepositoryAndSubmitReturn" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="getJobStatus">
  <complexType>
    <sequence>
      <element name="arg_0_7" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="getJobStatusResponse">
  <complexType>
    <sequence>
      <element name="getJobStatusReturn" type="xsd:int"/>
    </sequence>
  </complexType>
</element>
<element name="saveJobToRepositoryAndSubmit">
  <complexType>
    <sequence>
      <element name="arg_0_8" nillable="true" type="xsd:string"/>
      <element name="arg_1_8" nillable="true" type="xsd:string"/>
      <element name="arg_2_8" type="xsd:boolean"/>
    </sequence>
  </complexType>
</element>
<element name="saveJobToRepositoryAndSubmitResponse">
  <complexType>
    <sequence>
      <element name="saveJobToRepositoryAndSubmitReturn" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="resumeJob">
  <complexType>

```

```

    <sequence>
      <element name="arg_0_9" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="resumeJobResponse">
  <complexType>
    <sequence/>
  </complexType>
</element>
<element name="cancelRecurringRequest">
  <complexType>
    <sequence>
      <element name="arg_0_10" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>

<element name="cancelRecurringRequestResponse">
  <complexType>
    <sequence/>
  </complexType>
</element>
<element name="getBatchJobRC">
  <complexType>
    <sequence>
      <element name="arg_0_11" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="getBatchJobRCResponse">
  <complexType>
    <sequence>
      <element name="getBatchJobRCReturn" type="xsd:int"/>
    </sequence>
  </complexType>
</element>
<element name="showAllRecurringRequests">
  <complexType>
    <sequence/>
  </complexType>
</element>
<element name="showAllRecurringRequestsResponse">
  <complexType>
    <sequence>
      <element name="showAllRecurringRequestsReturn" nillable="true"
        type="impl:ArrayOf_xsd_nillable_string"/>
    </sequence>
  </complexType>
</element>
<element name="showJobFromRepository">
  <complexType>
    <sequence>
      <element name="arg_0_13" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="showJobFromRepositoryResponse">
  <complexType>
    <sequence>
      <element name="showJobFromRepositoryReturn" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="getJobOutput">
  <complexType>
    <sequence>
      <element name="arg_0_14" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="getJobOutputResponse">

```

```

    <complexType>
      <sequence>
        <element name="getJobOutputReturn" nillable="true" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="restartJob">
    <complexType>
      <sequence>
        <element name="arg_0_15" nillable="true" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="restartJobResponse">
    <complexType>
      <sequence/>
    </complexType>
  </element>
  <element name="getRecurringRequestDetails">
    <complexType>
      <sequence>
        <element name="arg_0_16" nillable="true" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="getRecurringRequestDetailsResponse">
    <complexType>
      <sequence>
        <element name="getRecurringRequestDetailsReturn" nillable="true" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="submitDelayedJob">
    <complexType>
      <sequence>
        <element name="arg_0_17" nillable="true" type="xsd:string"/>
        <element name="arg_1_17" nillable="true" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="submitDelayedJobResponse">
    <complexType>
      <sequence>
        <element name="submitDelayedJobReturn" nillable="true" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="submitDelayedJobFromRepository">
    <complexType>
      <sequence>
        <element name="arg_0_18" nillable="true" type="xsd:string"/>
        <element name="arg_1_18" nillable="true" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="submitDelayedJobFromRepositoryResponse">
    <complexType>
      <sequence>
        <element name="submitDelayedJobFromRepositoryReturn" nillable="true" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="cancelJob">
    <complexType>
      <sequence>
        <element name="arg_0_19" nillable="true" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="forcedCancelJob">
    <complexType>
      <sequence>

```

```

        <element name="arg_0_19" nillable="true" type="xsd:string"/>
    </sequence>
</complexType>
</element>
<element name="cancelJobResponse">
    <complexType>
        <sequence/>
    </complexType>
</element>
<element name="forcedCancelJobResponse">
    <complexType>
        <sequence/>
    </complexType>
</element>
<element name="submitRecurringRequestFromRepository">
    <complexType>
        <sequence>
            <element name="arg_0_20" nillable="true" type="xsd:string"/>
            <element name="arg_1_20" nillable="true" type="xsd:string"/>
            <element name="arg_2_20" nillable="true" type="xsd:string"/>
            <element name="arg_3_20" nillable="true" type="xsd:string"/>
        </sequence>
    </complexType>
</element>
<element name="submitRecurringRequestFromRepositoryResponse">
    <complexType>
        <sequence/>
    </complexType>
</element>
<element name="removeJobFromRepository">
    <complexType>
        <sequence>
            <element name="arg_0_21" nillable="true" type="xsd:string"/>
        </sequence>
    </complexType>
</element>
<element name="removeJobFromRepositoryResponse">
    <complexType>
        <sequence/>
    </complexType>
</element>
<element name="purgeJob">
    <complexType>
        <sequence>
            <element name="arg_0_22" nillable="true" type="xsd:string"/>
        </sequence>
    </complexType>
</element>
<element name="purgeJobResponse">
    <complexType>
        <sequence/>
    </complexType>
</element>
<element name="submitRecurringRequest">
    <complexType>
        <sequence>
            <element name="arg_0_23" nillable="true" type="xsd:string"/>
            <element name="arg_1_23" nillable="true" type="xsd:string"/>
            <element name="arg_2_23" nillable="true" type="xsd:string"/>
            <element name="arg_3_23" nillable="true" type="xsd:string"/>
        </sequence>
    </complexType>
</element>
<element name="submitRecurringRequestResponse">
    <complexType>
        <sequence/>
    </complexType>
</element>
<element name="showRecurringJobs">
    <complexType>
        <sequence>
            <element name="arg_0_24" nillable="true" type="xsd:string"/>
        </sequence>
    </complexType>
</element>

```



```

    </sequence>
  </complexType>
</element>
<element name="showRecurringJobsResponse">
  <complexType>
    <sequence>
      <element name="showRecurringJobsReturn" nillable="true"
        type="impl:ArrayOf_xsd_nillable_string"/>
    </sequence>
  </complexType>
</element>
<element name="getJobDetails">
  <complexType>
    <sequence>
      <element name="arg_0_25" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="getJobDetailsResponse">
  <complexType>
    <sequence>
      <element name="getJobDetailsReturn" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="submitJobFromRepository">
  <complexType>
    <sequence>
      <element name="arg_0_0" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="submitModifiableJobFromRepository">
  <complexType>
    <sequence>
      <element name="arg_0_26" nillable="true" type="xsd:string"/>
      <element name="arg_1_26" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="submitModifiableJobFromRepositoryResponse">
  <complexType>
    <sequence>
      <element name="submitModifiableJobFromRepositoryReturn" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="submitModifiableJob">
  <complexType>
    <sequence>
      <element name="arg_0_27" nillable="true" type="xsd:string"/>
      <element name="arg_1_27" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="submitModifiableJobResponse">
  <complexType>
    <sequence>
      <element name="submitModifiableJobReturn" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="saveModifiableDelayedJobToRepositoryAndSubmit">
  <complexType>
    <sequence>
      <element name="arg_0_28" nillable="true" type="xsd:string"/>
      <element name="arg_1_28" nillable="true" type="xsd:string"/>
      <element name="arg_2_28" type="xsd:boolean"/>
      <element name="arg_3_28" nillable="true" type="xsd:string"/>
      <element name="arg_4_28" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>

```

```

</element>
<element name="saveModifiableDelayedJobToRepositoryAndSubmitResponse">
  <complexType>
    <sequence>
      <element name="saveModifiableDelayedJobToRepositoryAndSubmitReturn" nillable="true"
        type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="saveModifiableJobToRepositoryAndSubmit">
  <complexType>
    <sequence>
      <element name="arg_0_29" nillable="true" type="xsd:string"/>
      <element name="arg_1_29" nillable="true" type="xsd:string"/>
      <element name="arg_2_29" type="xsd:boolean"/>
      <element name="arg_3_29" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="saveModifiableJobToRepositoryAndSubmitResponse">
  <complexType>
    <sequence>
      <element name="saveModifiableJobToRepositoryAndSubmitReturn" nillable="true"
        type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="submitModifiableDelayedJob">
  <complexType>
    <sequence>
      <element name="arg_0_30" nillable="true" type="xsd:string"/>
      <element name="arg_1_30" nillable="true" type="xsd:string"/>
      <element name="arg_2_30" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="submitModifiableDelayedJobResponse">
  <complexType>
    <sequence>
      <element name="submitModifiableDelayedJobReturn" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="submitModifiableDelayedJobFromRepository">
  <complexType>
    <sequence>
      <element name="arg_0_31" nillable="true" type="xsd:string"/>
      <element name="arg_1_31" nillable="true" type="xsd:string"/>
      <element name="arg_2_31" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="submitModifiableDelayedJobFromRepositoryResponse">
  <complexType>
    <sequence>
      <element name="submitModifiableDelayedJobFromRepositoryReturn" nillable="true"
        type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="submitModifiableRecurringRequestFromRepository">
  <complexType>
    <sequence>
      <element name="arg_0_32" nillable="true" type="xsd:string"/>
      <element name="arg_1_32" nillable="true" type="xsd:string"/>
      <element name="arg_2_32" nillable="true" type="xsd:string"/>
      <element name="arg_3_32" nillable="true" type="xsd:string"/>
      <element name="arg_4_32" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="submitModifiableRecurringRequestFromRepositoryResponse">

```

```

<complexType>
  <sequence/>
</complexType>
</element>
<element name="submitModifiableRecurringRequest">
  <complexType>
    <sequence>
      <element name="arg_0_33" nillable="true" type="xsd:string"/>
      <element name="arg_1_33" nillable="true" type="xsd:string"/>
      <element name="arg_2_33" nillable="true" type="xsd:string"/>
      <element name="arg_3_33" nillable="true" type="xsd:string"/>
      <element name="arg_4_33" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="submitModifiableRecurringRequestResponse">
  <complexType>
    <sequence/>
  </complexType>
</element>
<element name="modifyModifiableRecurringRequest">
  <complexType>
    <sequence>
      <element name="arg_0_34" nillable="true" type="xsd:string"/>
      <element name="arg_1_34" nillable="true" type="xsd:string"/>
      <element name="arg_2_34" nillable="true" type="xsd:string"/>
      <element name="arg_3_34" nillable="true" type="xsd:string"/>
      <element name="arg_4_34" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="stopJob">
  <complexType>
    <sequence>
      <element name="arg_0_35" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="getJobsName">
  <complexType>
    <sequence>
      <element name="arg_0_36" nillable="true" type="xsd:string"/>
      <element name="arg_1_36" nillable="true" type="xsd:string"/>
      <element name="arg_2_36" nillable="true" type="xsd:string"/>
      <element name="arg_3_36" type="xsd:boolean"/>
    </sequence>
  </complexType>
</element>
<element name="getSymbolicVariables">
  <complexType>
    <sequence>
      <element name="arg_0_37" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="getSymbolicVariablesResponse">
  <complexType>
    <sequence>
      <element name="getSymbolicVariablesReturn" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="getJobLog">
  <complexType>
    <sequence>
      <element name="arg_0_38" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="getJobLogResponse">
  <complexType>
    <sequence>

```

```

        <element name="getJobLogReturn" nillable="true" type="xsd:string"/>
    </sequence>
</complexType>
</element>
<element name="getLogMetaData">
    <complexType>
        <sequence>
            <element name="arg_0_39" nillable="true" type="xsd:string"/>
        </sequence>
    </complexType>
</element>
<element name="getLogMetaDataResponse">
    <complexType>
        <sequence>
            <element name="getLogMetaDataReturn" nillable="true" type="impl:ArrayOf_xsd_nillable_string"/>
        </sequence>
    </complexType>
</element>
<element name="getLogPartList">
    <complexType>
        <sequence>
            <element name="arg_0_40" nillable="true" type="xsd:string"/>
            <element name="arg_1_40" nillable="true" type="xsd:string"/>
        </sequence>
    </complexType>
</element>
<element name="getLogPartListResponse">
    <complexType>
        <sequence>
            <element name="getLogPartListReturn" nillable="true" type="impl:ArrayOf_xsd_nillable_string"/>
        </sequence>
    </complexType>
</element>
<element name="getLogPart">
    <complexType>
        <sequence>
            <element name="arg_0_41" nillable="true" type="xsd:string"/>
            <element name="arg_1_41" nillable="true" type="xsd:string"/>
            <element name="arg_2_41" nillable="true" type="xsd:string"/>
        </sequence>
    </complexType>
</element>
<element name="getLogPartResponse">
    <complexType>
        <sequence>
            <element name="getLogPartReturn" nillable="true" type="impl:ArrayOf_xsd_nillable_string"/>
        </sequence>
    </complexType>
</element>
<element name="getLogSize">
    <complexType>
        <sequence>
            <element name="arg_0_42" nillable="true" type="xsd:string"/>
            <element name="arg_1_42" nillable="true" type="xsd:string"/>
        </sequence>
    </complexType>
</element>
<element name="getLogSizeResponse">
    <complexType>
        <sequence>
            <element name="getLogSizeReturn" nillable="true" type="xsd:string"/>
        </sequence>
    </complexType>
</element>
<element name="getLogAge">
    <complexType>
        <sequence>
            <element name="arg_0_43" nillable="true" type="xsd:string"/>
            <element name="arg_1_43" nillable="true" type="xsd:string"/>
        </sequence>
    </complexType>
</element>

```

```

<element name="getLogAgeResponse">
  <complexType>
    <sequence>
      <element name="getLogAgeReturn" type="xsd:int"/>
    </sequence>
  </complexType>
</element>
<element name="getJobsByClass">
  <complexType>
    <sequence>
      <element name="arg_0_44" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="getJobsByClassResponse">
  <complexType>
    <sequence>
      <element name="getJobsByClassReturn" nillable="true"
        type="impl:ArrayOf_xsd_nillable_string"/>
    </sequence>
  </complexType>
</element>
<element name="removeJobLog">
  <complexType>
    <sequence>
      <element name="arg_0_45" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="removeJobLogResponse">
  <complexType>
    <sequence/>
  </complexType>
</element>
<element name="purgeJobLog">
  <complexType>
    <sequence>
      <element name="arg_0_46" nillable="true" type="xsd:string"/>
      <element name="arg_1_46" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="purgeJobLogResponse">
  <complexType>
    <sequence/>
  </complexType>
</element>
<element name="getAdminAddresses">
  <complexType>
    <sequence/>
  </complexType>
</element>
<element name="getAdminAddressesResponse">
  <complexType>
    <sequence>
      <element name="getAdminAddressesReturn" nillable="true"
        type="impl:ArrayOf_xsd_nillable_string"/>
    </sequence>
  </complexType>
</element>
<element name="getJobLogMetaDataByAgeForClass">
  <complexType>
    <sequence>
      <element name="arg_0_48" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="getJobLogMetaDataByAgeForClassResponse">
  <complexType>
    <sequence>
      <element name="getJobLogMetaDataByAgeForClassReturn" nillable="true"
        type="impl:ArrayOf_xsd_nillable_string"/>
    </sequence>
  </complexType>
</element>

```

```

    </sequence>
  </complexType>
</element>
<element name="getJobLogMetaDataBySizeForClass">
  <complexType>
    <sequence>
      <element name="arg_0_49" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="getJobLogMetaDataBySizeForClassResponse">
  <complexType>
    <sequence>
      <element name="getJobLogMetaDataBySizeForClassReturn" nillable="true"
        type="impl:ArrayOf_xsd_nillable_string"/>
    </sequence>
  </complexType>
</element>
<element name="quiesceLogging">
  <complexType>
    <sequence>
      <element name="arg_0_50" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="sendCheckpointNotification">
  <complexType>
    <sequence>
      <element name="arg_0_51" nillable="true" type="xsd:string"/>
      <element name="arg_1_51" nillable="true" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="sendCheckpointNotificationResponse">
  <complexType>
    <sequence/>
  </complexType>
</element>
<element name="quiesceLoggingResponse">
  <complexType>
    <sequence/>
  </complexType>
</element>
<element name="modifyModifiableRecurringRequestResponse">
  <complexType>
    <sequence/>
  </complexType>
</element>
<element name="stopJobResponse">
  <complexType>
    <sequence/>
  </complexType>
</element>
<element name="getJobsNameResponse">
  <complexType>
    <sequence/>
  </complexType>
</element>
<complexType name="InvalidJobNameException">
  <sequence>
    <element name="message" nillable="true" type="xsd:string"/>
  </sequence>
</complexType>
<element name="InvalidJobNameException" nillable="true" type="impl:InvalidJobNameException"/>
<complexType name="SchedulerException">
  <sequence>
    <element name="message" nillable="true" type="xsd:string"/>
  </sequence>
</complexType>
<element name="SchedulerException" nillable="true" type="impl:SchedulerException"/>
<complexType name="JCLException">
  <sequence>

```

```

    <element name="message" nillable="true" type="xsd:string"/>
  </sequence>
</complexType>
<element name="JCLEException" nillable="true" type="impl:JCLEException"/>
<complexType name="JobSubmissionException">
  <sequence>
    <element name="message" nillable="true" type="xsd:string"/>
  </sequence>
</complexType>
<element name="JobSubmissionException" nillable="true" type="impl:JobSubmissionException"/>
<complexType name="ArrayOf_xsd_nillable_string">
  <sequence>
    <element maxOccurs="unbounded" minOccurs="0" name="string" nillable="true"
      type="xsd:string"/>
  </sequence>
</complexType>
<complexType name="InvalidOperationException">
  <sequence>
    <element name="message" nillable="true" type="xsd:string"/>
  </sequence>
</complexType>
<element name="InvalidOperationException" nillable="true"
  type="impl:InvalidOperationException"/>
<complexType name="InvalidJobIDException">
  <sequence>
    <element name="message" nillable="true" type="xsd:string"/>
  </sequence>
</complexType>
<element name="InvalidJobIDException" nillable="true" type="impl:InvalidJobIDException"/>
<complexType name="InvalidStartDateTimeFormatException">
  <sequence>
    <element name="message" nillable="true" type="xsd:string"/>
  </sequence>
</complexType>
<element name="InvalidStartDateTimeFormatException" nillable="true"
  type="impl:InvalidStartDateTimeFormatException"/>
<complexType name="StaleTimeException">
  <sequence>
    <element name="message" nillable="true" type="xsd:string"/>
  </sequence>
</complexType>
<element name="StaleTimeException" nillable="true" type="impl:StaleTimeException"/>
<complexType name="InvalidIntervalException">
  <sequence>
    <element name="message" nillable="true" type="xsd:string"/>
  </sequence>
</complexType>
<element name="InvalidIntervalException" nillable="true" type="impl:InvalidIntervalException"/>
</schema>
</wsdl:types>

<wsdl:message name="showRecurringJobsRequest">
  <wsdl:part element="impl:showRecurringJobs" name="parameters"/>
</wsdl:message>

<wsdl:message name="cancelRecurringRequestResponse">
  <wsdl:part element="impl:cancelRecurringRequestResponse" name="parameters"/>
</wsdl:message>

<wsdl:message name="submitRecurringRequestRequest">
  <wsdl:part element="impl:submitRecurringRequest" name="parameters"/>
</wsdl:message>

<wsdl:message name="submitModifiableRecurringRequestRequest">
  <wsdl:part element="impl:submitModifiableRecurringRequest" name="parameters"/>

```

```

</wsdl:message>
<wsdl:message name="InvalidJobNameException">
  <wsdl:part element="impl:InvalidJobNameException" name="fault"/>
</wsdl:message>
<wsdl:message name="showAllJobsRequest">
  <wsdl:part element="impl:showAllJobs" name="parameters"/>
</wsdl:message>
<wsdl:message name="getRecurringRequestDetailsRequest">
  <wsdl:part element="impl:getRecurringRequestDetails" name="parameters"/>
</wsdl:message>
<wsdl:message name="getJobOutputRequest">
  <wsdl:part element="impl:getJobOutput" name="parameters"/>
</wsdl:message>
<wsdl:message name="InvalidStartDateTimeFormatException">
  <wsdl:part element="impl:InvalidStartDateTimeFormatException" name="fault"/>
</wsdl:message>
<wsdl:message name="resumeJobRequest">
  <wsdl:part element="impl:resumeJob" name="parameters"/>
</wsdl:message>
<wsdl:message name="saveDelayedJobToRepositoryAndSubmitRequest">
  <wsdl:part element="impl:saveDelayedJobToRepositoryAndSubmit" name="parameters"/>
</wsdl:message>
<wsdl:message name="saveModifiableDelayedJobToRepositoryAndSubmitRequest">
  <wsdl:part element="impl:saveModifiableDelayedJobToRepositoryAndSubmit" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitDelayedJobResponse">
  <wsdl:part element="impl:submitDelayedJobResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitModifiableDelayedJobResponse">
  <wsdl:part element="impl:submitModifiableDelayedJobResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="removeJobFromRepositoryResponse">
  <wsdl:part element="impl:removeJobFromRepositoryResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="StaleTimeException">
  <wsdl:part element="impl:StaleTimeException" name="fault"/>

```



```

</wsdl:message>
<wsdl:message name="getJobStatusResponse">
  <wsdl:part element="impl:getJobStatusResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="cancelRecurringRequestRequest">
  <wsdl:part element="impl:cancelRecurringRequest" name="parameters"/>
</wsdl:message>
<wsdl:message name="getBatchJobRCResponse">
  <wsdl:part element="impl:getBatchJobRCResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="getJobDetailsResponse">
  <wsdl:part element="impl:getJobDetailsResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitJobRequest">
  <wsdl:part element="impl:submitJob" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitModifiableJobRequest">
  <wsdl:part element="impl:submitModifiableJob" name="parameters"/>
</wsdl:message>
<wsdl:message name="purgeJobResponse">
  <wsdl:part element="impl:purgeJobResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="removeJobFromRepositoryRequest">
  <wsdl:part element="impl:removeJobFromRepository" name="parameters"/>
</wsdl:message>
<wsdl:message name="getJobOutputResponse">
  <wsdl:part element="impl:getJobOutputResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitJobFromRepositoryResponse">
  <wsdl:part element="impl:submitJobFromRepositoryResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitModifiableJobFromRepositoryResponse">
  <wsdl:part element="impl:submitModifiableJobFromRepositoryResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="JCLErrorException">
  <wsdl:part element="impl:JCLErrorException" name="fault"/>

```

```

</wsdl:message>
<wsdl:message name="cancelJobResponse">
  <wsdl:part element="impl:cancelJobResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="forcedCancelJobResponse">
  <wsdl:part element="impl:forcedCancelJobResponse" name="parameters"/>
</wsdl:message>

<wsdl:message name="submitDelayedJobRequest">
  <wsdl:part element="impl:submitDelayedJob" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitModifiableDelayedJobRequest">
  <wsdl:part element="impl:submitModifiableDelayedJob" name="parameters"/>
</wsdl:message>
<wsdl:message name="showAllJobsResponse">
  <wsdl:part element="impl:showAllJobsResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="showJobFromRepositoryRequest">
  <wsdl:part element="impl:showJobFromRepository" name="parameters"/>
</wsdl:message>
<wsdl:message name="JobSubmissionException">
  <wsdl:part element="impl:JobSubmissionException" name="fault"/>
</wsdl:message>
<wsdl:message name="showAllRecurringRequestsResponse">
  <wsdl:part element="impl:showAllRecurringRequestsResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="showAllRecurringRequestsRequest">
  <wsdl:part element="impl:showAllRecurringRequests" name="parameters"/>
</wsdl:message>
<wsdl:message name="saveDelayedJobToRepositoryAndSubmitResponse">
  <wsdl:part element="impl:saveDelayedJobToRepositoryAndSubmitResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="saveModifiableDelayedJobToRepositoryAndSubmitResponse">
  <wsdl:part element="impl:saveModifiableDelayedJobToRepositoryAndSubmitResponse"
    name="parameters"/>
</wsdl:message>
<wsdl:message name="suspendJobResponse">

```

```

        <wsdl:part element="impl:suspendJobResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitJobResponse">
    <wsdl:part element="impl:submitJobResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitModifiableJobResponse">
    <wsdl:part element="impl:submitModifiableJobResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="restartJobResponse">
    <wsdl:part element="impl:restartJobResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="saveJobToRepositoryAndSubmitRequest">
    <wsdl:part element="impl:saveJobToRepositoryAndSubmit" name="parameters"/>
</wsdl:message>
<wsdl:message name="saveModifiableJobToRepositoryAndSubmitRequest">
    <wsdl:part element="impl:saveModifiableJobToRepositoryAndSubmit" name="parameters"/>
</wsdl:message>
<wsdl:message name="restartJobRequest">
    <wsdl:part element="impl:restartJob" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitRecurringRequestResponse">
    <wsdl:part element="impl:submitRecurringRequestResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitModifiableRecurringRequestResponse">
    <wsdl:part element="impl:submitModifiableRecurringRequestResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="purgeJobRequest">
    <wsdl:part element="impl:purgeJob" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitDelayedJobFromRepositoryRequest">
    <wsdl:part element="impl:submitDelayedJobFromRepository" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitModifiableDelayedJobFromRepositoryRequest">
    <wsdl:part element="impl:submitModifiableDelayedJobFromRepository" name="parameters"/>
</wsdl:message>
<wsdl:message name="getSymbolicVariablesRequest">

```

```

    <wsdl:part element="impl:getSymbolicVariables" name="parameters"/>
</wsdl:message>
<wsdl:message name="cancelJobRequest">
    <wsdl:part element="impl:cancelJob" name="parameters"/>
</wsdl:message>
<wsdl:message name="forcedCancelJobRequest">
    <wsdl:part element="impl:forcedCancelJob" name="parameters"/>
</wsdl:message>

<wsdl:message name="getBatchJobRCRequest">
    <wsdl:part element="impl:getBatchJobRC" name="parameters"/>
</wsdl:message>
<wsdl:message name="getJobStatusRequest">
    <wsdl:part element="impl:getJobStatus" name="parameters"/>
</wsdl:message>
<wsdl:message name="modifyRecurringRequestResponse">
    <wsdl:part element="impl:modifyRecurringRequestResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="modifyModifiableRecurringRequestResponse">
    <wsdl:part element="impl:modifyModifiableRecurringRequestResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="stopJobResponse">
    <wsdl:part element="impl:stopJobResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="getJobsNameResponse">
    <wsdl:part element="impl:getJobsNameResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="saveJobToRepositoryResponse">
    <wsdl:part element="impl:saveJobToRepositoryResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="resumeJobResponse">
    <wsdl:part element="impl:resumeJobResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitRecurringRequestFromRepositoryResponse">
    <wsdl:part element="impl:submitRecurringRequestFromRepositoryResponse" name="parameters"/>
</wsdl:message>

```

```

<wsdl:message name="submitModifiableRecurringRequestFromRepositoryResponse">
  <wsdl:part element="impl:submitModifiableRecurringRequestFromRepositoryResponse"
    name="parameters"/>
</wsdl:message>
<wsdl:message name="submitRecurringRequestFromRepositoryRequest">
  <wsdl:part element="impl:submitRecurringRequestFromRepository" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitModifiableRecurringRequestFromRepositoryRequest">
  <wsdl:part element="impl:submitModifiableRecurringRequestFromRepository" name="parameters"/>
</wsdl:message>
<wsdl:message name="saveJobToRepositoryAndSubmitResponse">
  <wsdl:part element="impl:saveJobToRepositoryAndSubmitResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="saveModifiableJobToRepositoryAndSubmitResponse">
  <wsdl:part element="impl:saveModifiableJobToRepositoryAndSubmitResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitDelayedJobFromRepositoryResponse">
  <wsdl:part element="impl:submitDelayedJobFromRepositoryResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitModifiableDelayedJobFromRepositoryResponse">
  <wsdl:part element="impl:submitModifiableDelayedJobFromRepositoryResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="getSymbolicVariablesResponse">
  <wsdl:part element="impl:getSymbolicVariablesResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="SchedulerException">
  <wsdl:part element="impl:SchedulerException" name="fault"/>
</wsdl:message>
<wsdl:message name="InvalidOperationException">
  <wsdl:part element="impl:InvalidOperationException" name="fault"/>
</wsdl:message>
<wsdl:message name="InvalidIntervalException">
  <wsdl:part element="impl:InvalidIntervalException" name="fault"/>
</wsdl:message>
<wsdl:message name="showJobFromRepositoryResponse">
  <wsdl:part element="impl:showJobFromRepositoryResponse" name="parameters"/>
</wsdl:message>

```

```

<wsdl:message name="InvalidJobIDException">
  <wsdl:part element="impl:InvalidJobIDException" name="fault"/>
</wsdl:message>
<wsdl:message name="submitJobFromRepositoryRequest">
  <wsdl:part element="impl:submitJobFromRepository" name="parameters"/>
</wsdl:message>
<wsdl:message name="submitModifiableJobFromRepositoryRequest">
  <wsdl:part element="impl:submitModifiableJobFromRepository" name="parameters"/>
</wsdl:message>
<wsdl:message name="modifyRecurringRequestRequest">
  <wsdl:part element="impl:modifyRecurringRequest" name="parameters"/>
</wsdl:message>
<wsdl:message name="modifyModifiableRecurringRequestRequest">
  <wsdl:part element="impl:modifyModifiableRecurringRequest" name="parameters"/>
</wsdl:message>
<wsdl:message name="stopJobRequest">
  <wsdl:part element="impl:stopJob" name="parameters"/>
</wsdl:message>
<wsdl:message name="getJobsNameRequest">
  <wsdl:part element="impl:getJobsName" name="parameters"/>
</wsdl:message>
<wsdl:message name="suspendJobRequest">
  <wsdl:part element="impl:suspendJob" name="parameters"/>
</wsdl:message>
<wsdl:message name="saveJobToRepositoryRequest">
  <wsdl:part element="impl:saveJobToRepository" name="parameters"/>
</wsdl:message>
<wsdl:message name="getRecurringRequestDetailsResponse">
  <wsdl:part element="impl:getRecurringRequestDetailsResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="getJobDetailsRequest">
  <wsdl:part element="impl:getJobDetails" name="parameters"/>
</wsdl:message>
<wsdl:message name="showRecurringJobsResponse">
  <wsdl:part element="impl:showRecurringJobsResponse" name="parameters"/>
</wsdl:message>

```

```

<wsdl:message name="getJobLogRequest">
  <wsdl:part element="impl:getJobLog" name="parameters"/>
</wsdl:message>
<wsdl:message name="getJobLogResponse">
  <wsdl:part element="impl:getJobLogResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="getLogMetaDataRequest">
  <wsdl:part element="impl:getLogMetaData" name="parameters"/>
</wsdl:message>
<wsdl:message name="getLogMetaDataResponse">
  <wsdl:part element="impl:getLogMetaDataResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="getLogPartListRequest">
  <wsdl:part element="impl:getLogPartList" name="parameters"/>
</wsdl:message>
<wsdl:message name="getLogPartListResponse">
  <wsdl:part element="impl:getLogPartListResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="getLogPartRequest">
  <wsdl:part element="impl:getLogPart" name="parameters"/>
</wsdl:message>
<wsdl:message name="getLogPartResponse">
  <wsdl:part element="impl:getLogPartResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="getLogSizeRequest">
  <wsdl:part element="impl:getLogSize" name="parameters"/>
</wsdl:message>
<wsdl:message name="getLogSizeResponse">
  <wsdl:part element="impl:getLogSizeResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="getLogAgeRequest">
  <wsdl:part element="impl:getLogAge" name="parameters"/>
</wsdl:message>
<wsdl:message name="getLogAgeResponse">
  <wsdl:part element="impl:getLogAgeResponse" name="parameters"/>
</wsdl:message>

```

```

<wsdl:message name="getJobsByClassRequest">
  <wsdl:part element="impl:getJobsByClass" name="parameters"/>
</wsdl:message>
<wsdl:message name="getJobsByClassResponse">
  <wsdl:part element="impl:getJobsByClassResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="removeJobLogRequest">
  <wsdl:part element="impl:removeJobLog" name="parameters"/>
</wsdl:message>
<wsdl:message name="removeJobLogResponse">
  <wsdl:part element="impl:removeJobLogResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="purgeJobLogRequest">
  <wsdl:part element="impl:purgeJobLog" name="parameters"/>
</wsdl:message>
<wsdl:message name="purgeJobLogResponse">
  <wsdl:part element="impl:purgeJobLogResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="getAdminAddressesRequest">
  <wsdl:part element="impl:getAdminAddresses" name="parameters"/>
</wsdl:message>
<wsdl:message name="getAdminAddressesResponse">
  <wsdl:part element="impl:getAdminAddressesResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="getJobLogMetaDataByAgeForClassRequest">
  <wsdl:part element="impl:getJobLogMetaDataByAgeForClass" name="parameters"/>
</wsdl:message>
<wsdl:message name="getJobLogMetaDataByAgeForClassResponse">
  <wsdl:part element="impl:getJobLogMetaDataByAgeForClassResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="getJobLogMetaDataBySizeForClassRequest">
  <wsdl:part element="impl:getJobLogMetaDataBySizeForClass" name="parameters"/>
</wsdl:message>
<wsdl:message name="getJobLogMetaDataBySizeForClassResponse">
  <wsdl:part element="impl:getJobLogMetaDataBySizeForClassResponse" name="parameters"/>
</wsdl:message>

```



```

<wsdl:message name="quiesceLoggingRequest">
  <wsdl:part element="impl:quiesceLogging" name="parameters"/>
</wsdl:message>
<wsdl:message name="sendCheckpointNotificationRequest">
  <wsdl:part element="impl:sendCheckpointNotification" name="parameters"/>
</wsdl:message>
<wsdl:message name="quiesceLoggingResponse">
  <wsdl:part element="impl:quiesceLoggingResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="sendCheckpointNotificationResponse">
  <wsdl:part element="impl:sendCheckpointNotificationResponse" name="parameters"/>
</wsdl:message>
<wsdl:portType name="JobScheduler">
  <wsdl:operation name="submitJobFromRepository">
    <wsdl:input message="impl:submitJobFromRepositoryRequest"
      name="submitJobFromRepositoryRequest"/>
    <wsdl:output message="impl:submitJobFromRepositoryResponse"
      name="submitJobFromRepositoryResponse"/>
    <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
    <wsdl:fault message="impl:JobSubmissionException" name="JobSubmissionException"/>
    <wsdl:fault message="impl:JCLEException" name="JCLEException"/>
    <wsdl:fault message="impl:InvalidJobNameException" name="InvalidJobNameException"/>
  </wsdl:operation>
  <wsdl:operation name="submitJob">
    <wsdl:input message="impl:submitJobRequest" name="submitJobRequest"/>
    <wsdl:output message="impl:submitJobResponse" name="submitJobResponse"/>
    <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
    <wsdl:fault message="impl:JobSubmissionException" name="JobSubmissionException"/>
    <wsdl:fault message="impl:JCLEException" name="JCLEException"/>
  </wsdl:operation>
  <wsdl:operation name="showAllJobs">
    <wsdl:input message="impl:showAllJobsRequest" name="showAllJobsRequest"/>
    <wsdl:output message="impl:showAllJobsResponse" name="showAllJobsResponse"/>
    <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  </wsdl:operation>
  <wsdl:operation name="saveJobToRepository">
    <wsdl:input message="impl:saveJobToRepositoryRequest"

```

```

        name="saveJobToRepositoryRequest"/>
<wsdl:output message="impl:saveJobToRepositoryResponse"
    name="saveJobToRepositoryResponse"/>
<wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
<wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
<wsdl:fault message="impl:JCLException" name="JCLException"/>
</wsdl:operation>
<wsdl:operation name="suspendJob">
    <wsdl:input message="impl:suspendJobRequest" name="suspendJobRequest"/>
    <wsdl:output message="impl:suspendJobResponse" name="suspendJobResponse"/>
    <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
    <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
    <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="modifyRecurringRequest">
    <wsdl:input message="impl:modifyRecurringRequestRequest"
        name="modifyRecurringRequestRequest"/>
    <wsdl:output message="impl:modifyRecurringRequestResponse"
        name="modifyRecurringRequestResponse"/>
    <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
    <wsdl:fault message="impl:InvalidIntervalException" name="InvalidIntervalException"/>
    <wsdl:fault message="impl:StaleTimeException" name="StaleTimeException"/>
    <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
    <wsdl:fault message="impl:InvalidStartDateTimeFormatException"
        name="InvalidStartDateTimeFormatException"/>
    <wsdl:fault message="impl:JCLException" name="JCLException"/>
</wsdl:operation>
<wsdl:operation name="saveDelayedJobToRepositoryAndSubmit">
    <wsdl:input message="impl:saveDelayedJobToRepositoryAndSubmitRequest"
        name="saveDelayedJobToRepositoryAndSubmitRequest"/>
    <wsdl:output message="impl:saveDelayedJobToRepositoryAndSubmitResponse"
        name="saveDelayedJobToRepositoryAndSubmitResponse"/>
    <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
    <wsdl:fault message="impl:StaleTimeException" name="StaleTimeException"/>
    <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
    <wsdl:fault message="impl:JobSubmissionException" name="JobSubmissionException"/>
    <wsdl:fault message="impl:InvalidStartDateTimeFormatException"
        name="InvalidStartDateTimeFormatException"/>
    <wsdl:fault message="impl:JCLException" name="JCLException"/>
</wsdl:operation>

```

```

<wsdl:operation name="getJobStatus">
  <wsdl:input message="impl:getJobStatusRequest" name="getJobStatusRequest"/>
  <wsdl:output message="impl:getJobStatusResponse" name="getJobStatusResponse"/>
  <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="saveJobToRepositoryAndSubmit">
  <wsdl:input message="impl:saveJobToRepositoryAndSubmitRequest"
    name="saveJobToRepositoryAndSubmitRequest"/>
  <wsdl:output message="impl:saveJobToRepositoryAndSubmitResponse"
    name="saveJobToRepositoryAndSubmitResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  <wsdl:fault message="impl:JobSubmissionException" name="JobSubmissionException"/>
  <wsdl:fault message="impl:JCLException" name="JCLException"/>
</wsdl:operation>
<wsdl:operation name="resumeJob">
  <wsdl:input message="impl:resumeJobRequest" name="resumeJobRequest"/>
  <wsdl:output message="impl:resumeJobResponse" name="resumeJobResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="cancelRecurringRequest">
  <wsdl:input message="impl:cancelRecurringRequestRequest"
    name="cancelRecurringRequestRequest"/>
  <wsdl:output message="impl:cancelRecurringRequestResponse"
    name="cancelRecurringRequestResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="getBatchJobRC">
  <wsdl:input message="impl:getBatchJobRCRequest" name="getBatchJobRCRequest"/>
  <wsdl:output message="impl:getBatchJobRCResponse" name="getBatchJobRCResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>

```

```

<wsdl:operation name="showAllRecurringRequests">
  <wsdl:input message="impl:showAllRecurringRequestsRequest"
    name="showAllRecurringRequestsRequest"/>
  <wsdl:output message="impl:showAllRecurringRequestsResponse"
    name="showAllRecurringRequestsResponse"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="showJobFromRepository">
  <wsdl:input message="impl:showJobFromRepositoryRequest"
    name="showJobFromRepositoryRequest"/>
  <wsdl:output message="impl:showJobFromRepositoryResponse"
    name="showJobFromRepositoryResponse"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  <wsdl:fault message="impl:InvalidJobNameException" name="InvalidJobNameException"/>
</wsdl:operation>
<wsdl:operation name="getJobOutput">
  <wsdl:input message="impl:getJobOutputRequest" name="getJobOutputRequest"/>
  <wsdl:output message="impl:getJobOutputResponse" name="getJobOutputResponse"/>
  <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="restartJob">
  <wsdl:input message="impl:restartJobRequest" name="restartJobRequest"/>
  <wsdl:output message="impl:restartJobResponse" name="restartJobResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  <wsdl:fault message="impl:JobSubmissionException" name="JobSubmissionException"/>
  <wsdl:fault message="impl:JCLEException" name="JCLEException"/>
</wsdl:operation>
<wsdl:operation name="getRecurringRequestDetails">
  <wsdl:input message="impl:getRecurringRequestDetailsRequest"
    name="getRecurringRequestDetailsRequest"/>
  <wsdl:output message="impl:getRecurringRequestDetailsResponse"
    name="getRecurringRequestDetailsResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="submitDelayedJob">

```

```

<wsdl:input message="impl:submitDelayedJobRequest" name="submitDelayedJobRequest"/>
<wsdl:output message="impl:submitDelayedJobResponse" name="submitDelayedJobResponse"/>
<wsdl:fault message="impl:StaleTimeException" name="StaleTimeException"/>
<wsdl:fault message="impl:SchedulException" name="SchedulException"/>
<wsdl:fault message="impl:JobSubmissionException" name="JobSubmissionException"/>
<wsdl:fault message="impl:InvalidStartDateTimeFormatException"
    name="InvalidStartDateTimeFormatException"/>
<wsdl:fault message="impl:JCLEException" name="JCLEException"/>
</wsdl:operation>
<wsdl:operation name="submitDelayedJobFromRepository">
    <wsdl:input message="impl:submitDelayedJobFromRepositoryRequest"
        name="submitDelayedJobFromRepositoryRequest"/>
    <wsdl:output message="impl:submitDelayedJobFromRepositoryResponse"
        name="submitDelayedJobFromRepositoryResponse"/>
    <wsdl:fault message="impl:StaleTimeException" name="StaleTimeException"/>
    <wsdl:fault message="impl:SchedulException" name="SchedulException"/>
    <wsdl:fault message="impl:JobSubmissionException" name="JobSubmissionException"/>
    <wsdl:fault message="impl:InvalidStartDateTimeFormatException"
        name="InvalidStartDateTimeFormatException"/>
    <wsdl:fault message="impl:JCLEException" name="JCLEException"/>
    <wsdl:fault message="impl:InvalidJobNameException" name="InvalidJobNameException"/>
</wsdl:operation>
<wsdl:operation name="cancelJob">
    <wsdl:input message="impl:cancelJobRequest" name="cancelJobRequest"/>
    <wsdl:output message="impl:cancelJobResponse" name="cancelJobResponse"/>
    <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
    <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
    <wsdl:fault message="impl:SchedulException" name="SchedulException"/>
</wsdl:operation>
<wsdl:operation name="forcedCancelJob">
    <wsdl:input message="impl:forcedCancelJobRequest" name="forcedCancelJobRequest"/>
    <wsdl:output message="impl:forcedCancelJobResponse" name="forcedCancelJobResponse"/>
    <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
    <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
    <wsdl:fault message="impl:SchedulException" name="SchedulException"/>
</wsdl:operation>
<wsdl:operation name="submitRecurringRequestFromRepository">

```

```

<wsdl:input message="impl:submitRecurringRequestFromRepositoryRequest"
  name="submitRecurringRequestFromRepositoryRequest"/>
<wsdl:output message="impl:submitRecurringRequestFromRepositoryResponse"
  name="submitRecurringRequestFromRepositoryResponse"/>
<wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
<wsdl:fault message="impl:InvalidIntervalException" name="InvalidIntervalException"/>
<wsdl:fault message="impl:StaleTimeException" name="StaleTimeException"/>
<wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
<wsdl:fault message="impl:InvalidStartDateTimeFormatException"
  name="InvalidStartDateTimeFormatException"/>
<wsdl:fault message="impl:JCLEException" name="JCLEException"/>
<wsdl:fault message="impl:InvalidJobNameException" name="InvalidJobNameException"/>
</wsdl:operation>
<wsdl:operation name="removeJobFromRepository">
  <wsdl:input message="impl:removeJobFromRepositoryRequest"
    name="removeJobFromRepositoryRequest"/>
  <wsdl:output message="impl:removeJobFromRepositoryResponse"
    name="removeJobFromRepositoryResponse"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  <wsdl:fault message="impl:InvalidJobNameException" name="InvalidJobNameException"/>
</wsdl:operation>
<wsdl:operation name="purgeJob">
  <wsdl:input message="impl:purgeJobRequest" name="purgeJobRequest"/>
  <wsdl:output message="impl:purgeJobResponse" name="purgeJobResponse"/>
  <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="submitRecurringRequest">
  <wsdl:input message="impl:submitRecurringRequestRequest"
    name="submitRecurringRequestRequest"/>
  <wsdl:output message="impl:submitRecurringRequestResponse"
    name="submitRecurringRequestResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:InvalidIntervalException" name="InvalidIntervalException"/>
  <wsdl:fault message="impl:StaleTimeException" name="StaleTimeException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  <wsdl:fault message="impl:InvalidStartDateTimeFormatException"
    name="InvalidStartDateTimeFormatException"/>
  <wsdl:fault message="impl:JCLEException" name="JCLEException"/>
</wsdl:operation>

```

```

<wsdl:operation name="showRecurringJobs">
  <wsdl:input message="impl:showRecurringJobsRequest" name="showRecurringJobsRequest"/>
  <wsdl:output message="impl:showRecurringJobsResponse" name="showRecurringJobsResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="getJobDetails">
  <wsdl:input message="impl:getJobDetailsRequest" name="getJobDetailsRequest"/>
  <wsdl:output message="impl:getJobDetailsResponse" name="getJobDetailsResponse"/>
  <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="submitModifiableJobFromRepository">
  <wsdl:input message="impl:submitModifiableJobFromRepositoryRequest"
    name="submitModifiableJobFromRepositoryRequest"/>
  <wsdl:output message="impl:submitModifiableJobFromRepositoryResponse"
    name="submitModifiableJobFromRepositoryResponse"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  <wsdl:fault message="impl:JobSubmissionException" name="JobSubmissionException"/>
  <wsdl:fault message="impl:JCLException" name="JCLException"/>
  <wsdl:fault message="impl:InvalidJobNameException" name="InvalidJobNameException"/>
</wsdl:operation>
<wsdl:operation name="submitModifiableJob">
  <wsdl:input message="impl:submitModifiableJobRequest"
    name="submitModifiableJobRequest"/>
  <wsdl:output message="impl:submitModifiableJobResponse"
    name="submitModifiableJobResponse"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  <wsdl:fault message="impl:JobSubmissionException" name="JobSubmissionException"/>
  <wsdl:fault message="impl:JCLException" name="JCLException"/>
</wsdl:operation>
<wsdl:operation name="saveModifiableDelayedJobToRepositoryAndSubmit">
  <wsdl:input message="impl:saveModifiableDelayedJobToRepositoryAndSubmitRequest"
    name="saveModifiableDelayedJobToRepositoryAndSubmitRequest"/>
  <wsdl:output message="impl:saveModifiableDelayedJobToRepositoryAndSubmitResponse"
    name="saveModifiableDelayedJobToRepositoryAndSubmitResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:StaleTimeException" name="StaleTimeException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>

```

```

<wsdl:fault message="impl:JobSubmissionException" name="JobSubmissionException"/>
<wsdl:fault message="impl:InvalidStartDateTimeFormatException"
  name="InvalidStartDateTimeFormatException"/>
<wsdl:fault message="impl:JCLEException" name="JCLEException"/>
</wsdl:operation>
<wsdl:operation name="saveModifiableJobToRepositoryAndSubmit">
  <wsdl:input message="impl:saveModifiableJobToRepositoryAndSubmitRequest"
    name="saveModifiableJobToRepositoryAndSubmitRequest"/>
  <wsdl:output message="impl:saveModifiableJobToRepositoryAndSubmitResponse"
    name="saveModifiableJobToRepositoryAndSubmitResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  <wsdl:fault message="impl:JobSubmissionException" name="JobSubmissionException"/>
  <wsdl:fault message="impl:JCLEException" name="JCLEException"/>
</wsdl:operation>
<wsdl:operation name="submitModifiableDelayedJob">
  <wsdl:input message="impl:submitModifiableDelayedJobRequest"
    name="submitModifiableDelayedJobRequest"/>
  <wsdl:output message="impl:submitModifiableDelayedJobResponse"
    name="submitModifiableDelayedJobResponse"/>
  <wsdl:fault message="impl:StaleTimeException" name="StaleTimeException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  <wsdl:fault message="impl:JobSubmissionException" name="JobSubmissionException"/>
  <wsdl:fault message="impl:InvalidStartDateTimeFormatException"
    name="InvalidStartDateTimeFormatException"/>
  <wsdl:fault message="impl:JCLEException" name="JCLEException"/>
</wsdl:operation>
<wsdl:operation name="submitModifiableDelayedJobFromRepository">
  <wsdl:input message="impl:submitModifiableDelayedJobFromRepositoryRequest"
    name="submitModifiableDelayedJobFromRepositoryRequest"/>
  <wsdl:output message="impl:submitModifiableDelayedJobFromRepositoryResponse"
    name="submitModifiableDelayedJobFromRepositoryResponse"/>
  <wsdl:fault message="impl:StaleTimeException" name="StaleTimeException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  <wsdl:fault message="impl:JobSubmissionException" name="JobSubmissionException"/>
  <wsdl:fault message="impl:InvalidStartDateTimeFormatException"
    name="InvalidStartDateTimeFormatException"/>
  <wsdl:fault message="impl:JCLEException" name="JCLEException"/>
  <wsdl:fault message="impl:InvalidJobNameException" name="InvalidJobNameException"/>
</wsdl:operation>
<wsdl:operation name="submitModifiableRecurringRequestFromRepository">

```



```

<wsdl:input message="impl:submitModifiableRecurringRequestFromRepositoryRequest"
  name="submitModifiableRecurringRequestFromRepositoryRequest"/>
<wsdl:output message="impl:submitModifiableRecurringRequestFromRepositoryResponse"
  name="submitModifiableRecurringRequestFromRepositoryResponse"/>
<wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
<wsdl:fault message="impl:InvalidIntervalException" name="InvalidIntervalException"/>
<wsdl:fault message="impl:StaleTimeException" name="StaleTimeException"/>
<wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
<wsdl:fault message="impl:InvalidStartDateTimeFormatException"
  name="InvalidStartDateTimeFormatException"/>
<wsdl:fault message="impl:JCLEException" name="JCLEException"/>
<wsdl:fault message="impl:InvalidJobNameException" name="InvalidJobNameException"/>
</wsdl:operation>
<wsdl:operation name="submitModifiableRecurringRequest">
  <wsdl:input message="impl:submitModifiableRecurringRequestRequest"
    name="submitModifiableRecurringRequestRequest"/>
  <wsdl:output message="impl:submitModifiableRecurringRequestResponse"
    name="submitModifiableRecurringRequestResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:InvalidIntervalException" name="InvalidIntervalException"/>
  <wsdl:fault message="impl:StaleTimeException" name="StaleTimeException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  <wsdl:fault message="impl:InvalidStartDateTimeFormatException"
    name="InvalidStartDateTimeFormatException"/>
  <wsdl:fault message="impl:JCLEException" name="JCLEException"/>
</wsdl:operation>
<wsdl:operation name="modifyModifiableRecurringRequest">
  <wsdl:input message="impl:modifyModifiableRecurringRequestRequest"
    name="modifyModifiableRecurringRequestRequest"/>
  <wsdl:output message="impl:modifyModifiableRecurringRequestResponse"
    name="modifyModifiableRecurringRequestResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:InvalidIntervalException" name="InvalidIntervalException"/>
  <wsdl:fault message="impl:StaleTimeException" name="StaleTimeException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  <wsdl:fault message="impl:InvalidStartDateTimeFormatException"
    name="InvalidStartDateTimeFormatException"/>
  <wsdl:fault message="impl:JCLEException" name="JCLEException"/>
</wsdl:operation>
<wsdl:operation name="stopJob">

```

```

<wsdl:input message="impl:stopJobRequest" name="stopJobRequest"/>
<wsdl:output message="impl:stopJobResponse" name="stopJobResponse"/>
<wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
<wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
<wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="getJobsName">
  <wsdl:input message="impl:getJobsNameRequest" name="getJobsNameRequest"/>
  <wsdl:output message="impl:getJobsNameResponse" name="getJobsNameResponse"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="getSymbolicVariables">
  <wsdl:input message="impl:getSymbolicVariablesRequest"
    name="getSymbolicVariablesRequest"/>
  <wsdl:output message="impl:getSymbolicVariablesResponse"
    name="getSymbolicVariablesResponse"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
  <wsdl:fault message="impl:JCLEException" name="JCLEException"/>
</wsdl:operation>
<wsdl:operation name="getJobLog">
  <wsdl:input message="impl:getJobLogRequest" name="getJobLogRequest"/>
  <wsdl:output message="impl:getJobLogResponse" name="getJobLogResponse"/>
  <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="getLogMetaData">
  <wsdl:input message="impl:getLogMetaDataRequest" name="getLogMetaDataRequest"/>
  <wsdl:output message="impl:getLogMetaDataResponse" name="getLogMetaDataResponse"/>
  <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="getLogPartList">
  <wsdl:input message="impl:getLogPartListRequest" name="getLogPartListRequest"/>
  <wsdl:output message="impl:getLogPartListResponse" name="getLogPartListResponse"/>
  <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>

```

```

        <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
        <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
    </wsdl:operation>
    <wsdl:operation name="getLogPart">
        <wsdl:input message="impl:getLogPartRequest" name="getLogPartRequest"/>
        <wsdl:output message="impl:getLogPartResponse" name="getLogPartResponse"/>
        <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
        <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
        <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
    </wsdl:operation>
    <wsdl:operation name="getLogSize">
        <wsdl:input message="impl:getLogSizeRequest" name="getLogSizeRequest"/>
        <wsdl:output message="impl:getLogSizeResponse" name="getLogSizeResponse"/>
        <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
        <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
        <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
    </wsdl:operation>
    <wsdl:operation name="getLogAge">
        <wsdl:input message="impl:getLogAgeRequest" name="getLogAgeRequest"/>
        <wsdl:output message="impl:getLogAgeResponse" name="getLogAgeResponse"/>
        <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
        <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
        <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
    </wsdl:operation>
    <wsdl:operation name="getJobsByClass">
        <wsdl:input message="impl:getJobsByClassRequest" name="getJobsByClassRequest"/>
        <wsdl:output message="impl:getJobsByClassResponse" name="getJobsByClassResponse"/>
        <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
        <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
    </wsdl:operation>
    <wsdl:operation name="removeJobLog">
        <wsdl:input message="impl:removeJobLogRequest" name="removeJobLogRequest"/>
        <wsdl:output message="impl:removeJobLogResponse" name="removeJobLogResponse"/>
        <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
        <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
        <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>

```

```

</wsdl:operation>
<wsdl:operation name="purgeJobLog">
  <wsdl:input message="impl:purgeJobLogRequest" name="purgeJobLogRequest"/>
  <wsdl:output message="impl:purgeJobLogResponse" name="purgeJobLogResponse"/>
  <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="getAdminAddresses">
  <wsdl:input message="impl:getAdminAddressesRequest" name="getAdminAddressesRequest"/>
  <wsdl:output message="impl:getAdminAddressesResponse" name="getAdminAddressesResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="getJobLogMetaDataByAgeForClass">
  <wsdl:input message="impl:getJobLogMetaDataByAgeForClassRequest"
    name="getJobLogMetaDataByAgeForClassRequest"/>
  <wsdl:output message="impl:getJobLogMetaDataByAgeForClassResponse"
    name="getJobLogMetaDataByAgeForClassResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="getJobLogMetaDataBySizeForClass">
  <wsdl:input message="impl:getJobLogMetaDataBySizeForClassRequest"
    name="getJobLogMetaDataBySizeForClassRequest"/>
  <wsdl:output message="impl:getJobLogMetaDataBySizeForClassResponse"
    name="getJobLogMetaDataBySizeForClassResponse"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="quiesceLogging">
  <wsdl:input message="impl:quiesceLoggingRequest" name="quiesceLoggingRequest"/>
  <wsdl:output message="impl:quiesceLoggingResponse" name="quiesceLoggingResponse"/>
  <wsdl:fault message="impl:InvalidJobIDException" name="InvalidJobIDException"/>
  <wsdl:fault message="impl:InvalidOperationException" name="InvalidOperationException"/>
  <wsdl:fault message="impl:SchedulerException" name="SchedulerException"/>
</wsdl:operation>
<wsdl:operation name="sendCheckpointNotification">

```

```

        <wsdl:input message="impl:sendCheckpointNotificationRequest"
            name="sendCheckpointNotificationRequest"/>

        <wsdl:output message="impl:sendCheckpointNotificationResponse"
            name="sendCheckpointNotificationResponse"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="JobSchedulerSoapBinding" type="impl:JobScheduler">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="submitJobFromRepository">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="submitJobFromRepositoryRequest">
            <wsdlsoap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="submitJobFromRepositoryResponse">
            <wsdlsoap:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="SchedulerException">
            <wsdlsoap:fault name="SchedulerException" use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="JobSubmissionException">
            <wsdlsoap:fault name="JobSubmissionException" use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="JCLErrorException">
            <wsdlsoap:fault name="JCLErrorException" use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="InvalidJobNameException">
            <wsdlsoap:fault name="InvalidJobNameException" use="literal"/>
        </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="submitJob">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="submitJobRequest">
            <wsdlsoap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="submitJobResponse">
            <wsdlsoap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>

```

```

<wsdl:fault name="SchedulerException">
  <wsdlsoap:fault name="SchedulerException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="JobSubmissionException">
  <wsdlsoap:fault name="JobSubmissionException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="JCLEException">
  <wsdlsoap:fault name="JCLEException" use="literal"/>
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="showAllJobs">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="showAllJobsRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="showAllJobsResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="saveJobToRepository">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="saveJobToRepositoryRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="saveJobToRepositoryResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidOperationException">
    <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>

```

```

    <wsdl:fault name="JCLException">
      <wsdlsoap:fault name="JCLException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>
  <wsdl:operation name="suspendJob">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="suspendJobRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="suspendJobResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
      <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidJobIDException">
      <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
      <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>
  <wsdl:operation name="modifyRecurringRequest">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="modifyRecurringRequestRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="modifyRecurringRequestResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
      <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidIntervalException">
      <wsdlsoap:fault name="InvalidIntervalException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

```

```

<wsdl:fault name="StaleTimeException">
  <wsdlsoap:fault name="StaleTimeException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="SchedulerException">
  <wsdlsoap:fault name="SchedulerException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="JCLEException">
  <wsdlsoap:fault name="JCLEException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="InvalidStartDateTimeFormatException">
  <wsdlsoap:fault name="InvalidStartDateTimeFormatException" use="literal"/>
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="saveDelayedJobToRepositoryAndSubmit">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="saveDelayedJobToRepositoryAndSubmitRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="saveDelayedJobToRepositoryAndSubmitResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidOperationException">
    <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="StaleTimeException">
    <wsdlsoap:fault name="StaleTimeException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="JobSubmissionException">
    <wsdlsoap:fault name="JobSubmissionException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="JCLEException">
    <wsdlsoap:fault name="JCLEException" use="literal"/>
  </wsdl:fault>

```



```

    <wsdl:fault name="InvalidStartDateTimeFormatException">
      <wsdlsoap:fault name="InvalidStartDateTimeFormatException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>
  <wsdl:operation name="getJobStatus">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getJobStatusRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getJobStatusResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidJobIDException">
      <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
      <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>
  <wsdl:operation name="saveJobToRepositoryAndSubmit">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="saveJobToRepositoryAndSubmitRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="saveJobToRepositoryAndSubmitResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
      <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
      <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="JobSubmissionException">
      <wsdlsoap:fault name="JobSubmissionException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

```

```

    <wsdl:fault name="JCLEException">
      <wsdlsoap:fault name="JCLEException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>
  <wsdl:operation name="resumeJob">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="resumeJobRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="resumeJobResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
      <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidJobIDException">
      <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
      <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>
  <wsdl:operation name="cancelRecurringRequest">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="cancelRecurringRequestRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="cancelRecurringRequestResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
      <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
      <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

```

```

</wsdl:operation>
<wsdl:operation name="getBatchJobRC">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="getBatchJobRCRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="getBatchJobRCResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidOperationException">
    <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidJobIDException">
    <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="showAllRecurringRequests">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="showAllRecurringRequestsRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="showAllRecurringRequestsResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="showJobFromRepository">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="showJobFromRepositoryRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>

```

```

<wsdl:output name="showJobFromRepositoryResponse">
  <wsdlsoap:body use="literal"/>
</wsdl:output>
<wsdl:fault name="SchedulerException">
  <wsdlsoap:fault name="SchedulerException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="InvalidJobNameException">
  <wsdlsoap:fault name="InvalidJobNameException" use="literal"/>
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="getJobOutput">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="getJobOutputRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="getJobOutputResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidJobIDException">
    <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="restartJob">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="restartJobRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="restartJobResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidOperationException">
    <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
  </wsdl:fault>

```

```

<wsdl:fault name="InvalidJobIDException">
  <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="SchedulerException">
  <wsdlsoap:fault name="SchedulerException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="JobSubmissionException">
  <wsdlsoap:fault name="JobSubmissionException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="JCLEException">
  <wsdlsoap:fault name="JCLEException" use="literal"/>
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="getRecurringRequestDetails">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="getRecurringRequestDetailsRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="getRecurringRequestDetailsResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidOperationException">
    <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="submitDelayedJob">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="submitDelayedJobRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="submitDelayedJobResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>

```

```

<wsdl:fault name="StaleTimeException">
  <wsdlsoap:fault name="StaleTimeException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="SchedulerException">
  <wsdlsoap:fault name="SchedulerException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="JobSubmissionException">
  <wsdlsoap:fault name="JobSubmissionException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="JCLException">
  <wsdlsoap:fault name="JCLException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="InvalidStartDateTimeFormatException">
  <wsdlsoap:fault name="InvalidStartDateTimeFormatException" use="literal"/>
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="submitDelayedJobFromRepository">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="submitDelayedJobFromRepositoryRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="submitDelayedJobFromRepositoryResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="StaleTimeException">
    <wsdlsoap:fault name="StaleTimeException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="JobSubmissionException">
    <wsdlsoap:fault name="JobSubmissionException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="JCLException">
    <wsdlsoap:fault name="JCLException" use="literal"/>
  </wsdl:fault>

```

```

<wsdl:fault name="InvalidStartDateTimeFormatException">
  <wsdlsoap:fault name="InvalidStartDateTimeFormatException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="InvalidJobNameException">
  <wsdlsoap:fault name="InvalidJobNameException" use="literal"/>
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="cancelJob">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="cancelJobRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="cancelJobResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidOperationException">
    <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidJobIDException">
    <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="forcedCancelJob">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="forcedCancelJobRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="forcedCancelJobResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidOperationException">
    <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
  </wsdl:fault>

```

```

<wsdl:fault name="InvalidJobIDException">
  <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="SchedulerException">
  <wsdlsoap:fault name="SchedulerException" use="literal"/>
</wsdl:fault>
</wsdl:operation>

<wsdl:operation name="submitRecurringRequestFromRepository">
  <wsdlsoap:operation soapAction="" />
  <wsdl:input name="submitRecurringRequestFromRepositoryRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="submitRecurringRequestFromRepositoryResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidOperationException">
    <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidIntervalException">
    <wsdlsoap:fault name="InvalidIntervalException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="StaleTimeException">
    <wsdlsoap:fault name="StaleTimeException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="JCLEException">
    <wsdlsoap:fault name="JCLEException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidStartDateTimeFormatException">
    <wsdlsoap:fault name="InvalidStartDateTimeFormatException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidJobNameException">
    <wsdlsoap:fault name="InvalidJobNameException" use="literal"/>
  </wsdl:fault>

```



```

</wsdl:operation>
<wsdl:operation name="removeJobFromRepository">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="removeJobFromRepositoryRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="removeJobFromRepositoryResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidJobNameException">
    <wsdlsoap:fault name="InvalidJobNameException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="purgeJob">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="purgeJobRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="purgeJobResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidJobIDException">
    <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="submitRecurringRequest">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="submitRecurringRequestRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>

```

```

<wsdl:output name="submitRecurringRequestResponse">
  <wsdlsoap:body use="literal"/>
</wsdl:output>
<wsdl:fault name="InvalidOperationException">
  <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="InvalidIntervalException">
  <wsdlsoap:fault name="InvalidIntervalException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="StaleTimeException">
  <wsdlsoap:fault name="StaleTimeException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="SchedulerException">
  <wsdlsoap:fault name="SchedulerException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="JCLException">
  <wsdlsoap:fault name="JCLException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="InvalidStartDateTimeFormatException">
  <wsdlsoap:fault name="InvalidStartDateTimeFormatException" use="literal"/>
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="showRecurringJobs">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="showRecurringJobsRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="showRecurringJobsResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidOperationException">
    <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>

```

```

</wsdl:operation>

<wsdl:operation name="getJobDetails">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="getJobDetailsRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="getJobDetailsResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidJobIDException">
    <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="submitModifiableJobFromRepository">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="submitModifiableJobFromRepositoryRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="submitModifiableJobFromRepositoryResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="JobSubmissionException">
    <wsdlsoap:fault name="JobSubmissionException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="JCLException">
    <wsdlsoap:fault name="JCLException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidJobNameException">
    <wsdlsoap:fault name="InvalidJobNameException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="submitModifiableJob">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="submitModifiableJobRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="submitModifiableJobResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="JobSubmissionException">
    <wsdlsoap:fault name="JobSubmissionException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="JCLException">
    <wsdlsoap:fault name="JCLException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="saveModifiableDelayedJobToRepositoryAndSubmit">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="saveModifiableDelayedJobToRepositoryAndSubmitRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="saveModifiableDelayedJobToRepositoryAndSubmitResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidOperationException">
    <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="StaleTimeException">

```

```

        <wsdlsoap:fault name="StaleTimeException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
        <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="JobSubmissionException">
        <wsdlsoap:fault name="JobSubmissionException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="JCLException">
        <wsdlsoap:fault name="JCLException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidStartDateTimeFormatException">
        <wsdlsoap:fault name="InvalidStartDateTimeFormatException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="saveModifiableJobToRepositoryAndSubmit">
    <wsdlsoap:operation soapAction="" />
    <wsdl:input name="saveModifiableJobToRepositoryAndSubmitRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="saveModifiableJobToRepositoryAndSubmitResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
        <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
        <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="JobSubmissionException">
        <wsdlsoap:fault name="JobSubmissionException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="JCLException">
        <wsdlsoap:fault name="JCLException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="submitModifiableDelayedJob">
    <wsdlsoap:operation soapAction="" />
    <wsdl:input name="submitModifiableDelayedJobRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="submitModifiableDelayedJobResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="StaleTimeException">
        <wsdlsoap:fault name="StaleTimeException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
        <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="JobSubmissionException">
        <wsdlsoap:fault name="JobSubmissionException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="JCLException">
        <wsdlsoap:fault name="JCLException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidStartDateTimeFormatException">
        <wsdlsoap:fault name="InvalidStartDateTimeFormatException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="submitModifiableDelayedJobFromRepository">
    <wsdlsoap:operation soapAction="" />
    <wsdl:input name="submitModifiableDelayedJobFromRepositoryRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="submitModifiableDelayedJobFromRepositoryResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="StaleTimeException">

```

```

        <wsdlsoap:fault name="StaleTimeException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
        <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="JobSubmissionException">
        <wsdlsoap:fault name="JobSubmissionException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="JCLException">
        <wsdlsoap:fault name="JCLException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidStartDateTimeFormatException">
        <wsdlsoap:fault name="InvalidStartDateTimeFormatException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidJobNameException">
        <wsdlsoap:fault name="InvalidJobNameException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="submitModifiableRecurringRequestFromRepository">
    <wsdlsoap:operation soapAction="" />
    <wsdl:input name="submitModifiableRecurringRequestFromRepositoryRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="submitModifiableRecurringRequestFromRepositoryResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
        <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidIntervalException">
        <wsdlsoap:fault name="InvalidIntervalException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="StaleTimeException">
        <wsdlsoap:fault name="StaleTimeException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
        <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="JCLException">
        <wsdlsoap:fault name="JCLException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidStartDateTimeFormatException">
        <wsdlsoap:fault name="InvalidStartDateTimeFormatException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidJobNameException">
        <wsdlsoap:fault name="InvalidJobNameException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="submitModifiableRecurringRequest">
    <wsdlsoap:operation soapAction="" />
    <wsdl:input name="submitModifiableRecurringRequestRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="submitModifiableRecurringRequestResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
        <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidIntervalException">
        <wsdlsoap:fault name="InvalidIntervalException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="StaleTimeException">
        <wsdlsoap:fault name="StaleTimeException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
        <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="JCLException">
        <wsdlsoap:fault name="JCLException" use="literal"/>
    </wsdl:fault>

```

```

    </wsdl:fault>
    <wsdl:fault name="InvalidStartDateTimeFormatException">
      <wsdlsoap:fault name="InvalidStartDateTimeFormatException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

  <wsdl:operation name="modifyModifiableRecurringRequest">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="modifyModifiableRecurringRequestRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="modifyModifiableRecurringRequestResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
      <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidIntervalException">
      <wsdlsoap:fault name="InvalidIntervalException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="StaleTimeException">
      <wsdlsoap:fault name="StaleTimeException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
      <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="JCLException">
      <wsdlsoap:fault name="JCLException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidStartDateTimeFormatException">
      <wsdlsoap:fault name="InvalidStartDateTimeFormatException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

  <wsdl:operation name="stopJob">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="stopJobRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="stopJobResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
      <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidJobIDException">
      <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
      <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

  <wsdl:operation name="getJobsName">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getJobsNameRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getJobsNameResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="SchedulerException">
      <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

  <wsdl:operation name="getSymbolicVariables">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getSymbolicVariablesRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getSymbolicVariablesResponse">

```

```

        <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="SchedulerException">
        <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="JCLEException">
        <wsdlsoap:fault name="JCLEException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="getJobLog">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getJobLogRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getJobLogResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
        <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidJobIDException">
        <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
        <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="getLogMetaData">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getLogMetaDataRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getLogMetaDataResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
        <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidJobIDException">
        <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
        <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="getLogPartList">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getLogPartListRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getLogPartListResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
        <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidJobIDException">
        <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
        <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="getLogPart">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getLogPartRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>

```

```

    <wsdl:output name="getLogPartResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
      <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidJobIDException">
      <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
      <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

  <wsdl:operation name="getLogSize">
    <wsdlsoap:operation soapAction="" />
    <wsdl:input name="getLogSizeRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getLogSizeResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
      <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidJobIDException">
      <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
      <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

  <wsdl:operation name="getLogAge">
    <wsdlsoap:operation soapAction="" />
    <wsdl:input name="getLogAgeRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getLogAgeResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
      <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidJobIDException">
      <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
      <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

  <wsdl:operation name="getJobsByClass">
    <wsdlsoap:operation soapAction="" />
    <wsdl:input name="getJobsByClassRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getJobsByClassResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
      <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
      <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>

  <wsdl:operation name="removeJobLog">
    <wsdlsoap:operation soapAction="" />
    <wsdl:input name="removeJobLogRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
  </wsdl:operation>

```



```

</wsdl:input>
<wsdl:output name="removeJobLogResponse">
  <wsdlsoap:body use="literal"/>
</wsdl:output>
<wsdl:fault name="InvalidOperationException">
  <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="InvalidJobIDException">
  <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
</wsdl:fault>
<wsdl:fault name="SchedulerException">
  <wsdlsoap:fault name="SchedulerException" use="literal"/>
</wsdl:fault>
</wsdl:operation>

<wsdl:operation name="purgeJobLog">
  <wsdlsoap:operation soapAction="" />
  <wsdl:input name="purgeJobLogRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="purgeJobLogResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidOperationException">
    <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidJobIDException">
    <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="getAdminAddresses">
  <wsdlsoap:operation soapAction="" />
  <wsdl:input name="getAdminAddressesRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="getAdminAddressesResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidOperationException">
    <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="getJobLogMetaDataByAgeForClass">
  <wsdlsoap:operation soapAction="" />
  <wsdl:input name="getJobLogMetaDataByAgeForClassRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="getJobLogMetaDataByAgeForClassResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="InvalidOperationException">
    <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
  </wsdl:fault>
  <wsdl:fault name="SchedulerException">
    <wsdlsoap:fault name="SchedulerException" use="literal"/>
  </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="getJobLogMetaDataBySizeForClass">
  <wsdlsoap:operation soapAction="" />
  <wsdl:input name="getJobLogMetaDataBySizeForClassRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="getJobLogMetaDataBySizeForClassResponse">

```

```

        <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
        <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
        <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="quiesceLogging">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="quiesceLoggingRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="quiesceLoggingResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="InvalidOperationException">
        <wsdlsoap:fault name="InvalidOperationException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidJobIDException">
        <wsdlsoap:fault name="InvalidJobIDException" use="literal"/>
    </wsdl:fault>
    <wsdl:fault name="SchedulerException">
        <wsdlsoap:fault name="SchedulerException" use="literal"/>
    </wsdl:fault>
</wsdl:operation>

<wsdl:operation name="sendCheckpointNotification">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="sendCheckpointNotificationRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="sendCheckpointNotificationResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>

</wsdl:binding>

<wsdl:service name="JobSchedulerService">
    <wsdl:port binding="impl:JobSchedulerSoapBinding" name="JobScheduler">
        <wsdlsoap:address
location="http://localhost:9080/LongRunningJobSchedulerWebSvcRouter/services/JobScheduler"/>
    </wsdl:port>
</wsdl:service>

</wsdl:definitions>

```

Submitting jobs from an external job scheduler

You can submit jobs from an external job scheduler to batch using the WSGrid utility.

Before you begin

Set up the external scheduler interface.

About this task

Distributed operating systems **IBM i** If an external interface is configured to use the default messaging system, you can use the WSGrid command line script to submit a job to the job scheduler.

z/OS If an external interface is configured to use the default messaging system, you can use the WSGrid command line script or JZOS Batch Toolkit for z/OS SDKs to submit a job to the job scheduler. If an external interface is configured to use WebSphere MQ, you can use JCL to submit a job to the scheduler.

Procedure

- If an external interface is configured to use the default messaging system, use the WSGrid.shlbat command-line script to submit a job to the scheduler.
 1. Prepare the job properties file and the control properties file to use as input for WSGrid.shlbat.
 2. Run the WSGrid.shlbat script located in the `app_install_root/stack_products/WCG/bin` directory.
- **z/OS** If an external interface is configured to use the default messaging system, use JZOS Batch Toolkit for z/OS SDKs to submit the job to the scheduler using the WSGrid utility.
 1. Update the relevant fields in the WSGrid JCL template to use with JZOS Batch Toolkit for z/OS SDKs.
 2. Enter the z/OS submit command to submit the JCL.
- **z/OS** If an external interface is configured to use WebSphere MQ, use JCL to submit a job to the scheduler.
 1. Update the relevant fields in the WSGrid job template.
 2. Enter the submit command to submit the JCL.

Results

You have submitted a job from an external job scheduler to batch using the WSGrid utility.

What to do next

Review the output of the WSGrid utility.

WSGrid command-line utility

The WSGrid utility is a client application of the job scheduler message-driven interface. Use the WSGrid utility to facilitate control of WebSphere batch jobs by external workload schedulers, such as Tivoli Workload Scheduler.

The WSGrid utility is invoked by an external workload scheduler as part of a larger task choreography. The WSGrid utility submits a WebSphere batch job, then waits for its completion. The job log from the batch job is written to the standard output stream of the environment in which the WSGrid utility was invoked.

WSGrid supports three distinct ways of specifying a batch job. You can specify:

1. An xJCL file.
2. The name of an xJCL file in the job repository.
3. A set of simple properties that describe the batch job.

Invocation

Use the following syntax for invoking the WSGrid utility:

Syntax:

- `WSGrid{.bat|.sh}<job properties file>` - The job properties file is a fully qualified path name to the file containing the WSGrid job properties. If specified alone, it must also contain WSGrid control properties.
- `WSGrid{.bat|.sh}<control properties file><job properties file>` - The control properties file is a fully qualified path name to the file containing the WSGrid control properties.
- `WSGrid{.bat|.sh}<control properties file><xJCL file>` - The xJCL file is a fully qualified path name to the file containing an xJCL job definition.

WSGrid job template

The WSGrid job template is an input to the WSGrid utility. The template contains the properties that the WSGrid utility uses to interact with a batch job.

- “WSGrid job template”
- “Example JCL to submit a new job” on page 251
- “Example JCL to restart a job” on page 252

WSGrid job template

```
//WSGRIDT JOB ( ),MSGCLASS=H
//RUN EXEC PGM=WSGRID
//*****
//*
//* WSGrid Job Template
//*
//* WSGrid is a Java utility that runs under control of JZOS for the
//* purpose of sub-dispatching a batch job. The utility submits the
//* batch job, writes the batch joblog to STDOUT DD as the WCG job runs
//* and then ends with a return code indicating the RC of the batch job.
//*
//* RC codes
//* -----
//*
//* 0-3040 - user range
//*
//* This is a user defined return code returned by a completed
//* WCG job.
//*
//* 3041-4096 - system ran
//*
//* This is a return codes set by system prior to completion of
//* a WCG job. The following return codes are defined:
//*
//* 4084 (-12) - WCG job ended in restartable state
//* 4080 (-16) - fatal system error
//*
//*****
//*
//* Required settings:
//*
//* 1) WGCNTL symbol must be set to the path-qualified file
//* containing the WSGRID control properties. See comment
//* block near WGCNTL DD for further information.
//*
//* 2) WGSUB symbol must be set to the path-qualified file
//* containing a WCG xJCL job definition.
//*
//* 3) WAS_HOME environment variable must be set to name of
//* WAS_home directory. Find this variable under the STDENV DD.
//*
//* Optional settings:
//*
//* 1) WGSUBS symbol is set to the path-qualified file
//* containing WCG job substitution properties.
//*
//* 2) WGRSTRT symbol is set to the path-qualified file
//* used for restart processing output/input. See
//* comment block near WGRSTRT DD for further information.
//*
//*****
// SET WGCNTL=<path-qualified file name>
// SET WGJOB=<path-qualified file name>
//*SET WGSUBS=<path-qualified file name>
//*SET WGRSTRT=<path-qualified file name>
//*****
//*
//* WSGrid Control Properties - required DD
//*
//* queue-manager-name=<MQ queue manager name>
//* scheduler-input-queue=<MQ input queue as defined during configuration>
```

```

/** scheduler-output-queue=<MQ output queue as defined during configuration>
/** timeout=<in milliseconds for WSGRID to wait for next outputmessage from job scheduler>
/** debug=<true|false>
/**
/*******
//WGCNTL DD PATH='&WGCNTL.'
/**
/*******
/**
/** WSGrid Job Definition - required DD
/**
/** File must contain valid WCG JCL job definition.
/**
/*******
//WGJOB DD PATH='&WGJOB.'
/**
/*******
/**
/** WSGrid Job Substitutions - optional DD
/**
/** Values are of form:
/**
/** substitution-prop.<property name>=<property value>
/**
/*******
/**WGSUBS DD PATH='&WGSUBS.'
/**
/*******
/**
/** WSGrid Restart Token - optional DD
/**
/** Note: restart token is written if and only if this job step ends
/** with RC=4084 (-12).
/**
/** The WGRSTRT DD takes precedence over the WGJOB DD, so
/** if this job is restarted with a non-empty restart file
/** the WCG job identified by the token will be restarted - a
/** new job instance based on the job definition in WGJOB is
/** not created.
/**
/** restart-job=<job_id of the job to restart. specify only when restart a job>
/*******
/**WGRSTRT DD PATH='&WGRSTRT.'
/**          PATHOPTS=(ORDWR,OCREAT),
/**          PATHMODE=(SIRUSR,SIWUSR)
/**
/*******

```

Example JCL to submit a new job

Use the following example when you want to submit a job through the WSGrid utility.

```

//WSGRIDCI JOB 1,'HUTCH',MSGCLASS=0
//SUBMIT EXEC PGM=WSGRID,REGION=0M
//STEPLIB DD DSN=B7CELL.WSGRID.LOAD,DISP=SHR
// DD DISP=SHR,DSN=SYS1.MQM.SCSQLOAD
// DD DISP=SHR,DSN=SYS1.MQM.SCSQAUTH
/*******
//SYSPRINT DD SYSOUT=*
/*******
//WGCNTL DD *
queue-manager-name=MQW1
scheduler-input-queue=WASIQCG
scheduler-output-queue=WASOQCG
timeout=5000
submit-timeout=30000
//WGJOB DD PATH='/u/hutch/cg/jcl/SimpleCIxJCL_Ebc.xml'
//WGSUBS DD *
substitution-prop.calctime=5
/**

```

Example JCL to restart a job

Use the following example when you want to restart a job through the WSGrid utility.

```
//B7WSRST4 JOB 1,'Restart IVT',MSGCLASS=0,NOTIFY=?
//SUBMIT EXEC PGM=WSGRID,REGION=0M
//STEPLIB DD DSN=B7CELL.WSGRID.LOAD,DISP=SHR
// DD DISP=SHR,DSN=SYS1.MQM.SCSQLOAD
// DD DISP=SHR,DSN=SYS1.MQM.SCSQAUTH
//SYSPRINT DD SYSOUT=*
//WGCNTL DD *
queue-manager-name=MQW1
scheduler-input-queue=WASIQ
scheduler-output-queue=WASOQ
timeout=5000
restart-job=XDCGIVTtxt-Long:00004
//WGJOB DD PATH='/u/hutch/cg/jcl/EBCxJCL-XDCGIVT-Text_Long.xml'
```

WSGrid JCL template to use with JZOS Batch Toolkit for z/OS SDKs

The WSGrid job template is an input to the WSGrid utility. Use the WSGrid utility to submit a job to the batch scheduler that has an external interface configured to use the default messaging system. Use the template with JZOS Batch Toolkit for z/OS software development kits.

```
//WSGRIDT JOB (),MSGCLASS=H
//*****
//*
//* WSGrid Job Template
//*
//* WSGrid is a batch Java utility that runs under control of JZOS
//* for the purpose of sub-dispatching a WCG job. The utility submits
//* the WCG job, writes the WCG joblog to STDOUT DD as the WCG job runs
//* and then ends with a return code indicating the RC of the WCG job.
//*
//* RC codes
//* -----
//*
//* 0-3040 - user range
//*
//* This is a user defined return code returned by a completed
//* WCG job.
//*
//* 3041-4096 - system ran
//*
//* This is a return codes set by system prior to completion of
//* a WCG job. The following return codes are defined:
//*
//* 4084 (-12) - WCG job ended in restartable state
//* 4080 (-16) - fatal system error
//*
//*****
//*
//* Required settings:
//*
//* 1) WGCNTL symbol must be set to the path-qualified file
//* containing the WSGRID control properties. See comment
//* block near WGCNTL DD for further information.
//*
//* 2) WGSUB symbol must be set to the path-qualified file
//* containing a WCG xJCL job definition.
//*
//* 3) WAS_HOME environment variable must be set to name of
//* WAS_home directory. Find this variable under the STDENV DD.
//*
//* Optional settings:
//*
//* 1) WGSUBS symbol is set to the path-qualified file
//* containing WCG job substitution properties.
//*
//* 2) WGRSTRT symbol is set to the path-qualified file
//* used for restart processing output/input. See
//* comment block near WGRSTRT DD for further information.
//*
//*****
```

```

// SET WGCNTL=<path-qualified file name>
// SET WGJOB=<path-qualified file name>
//*SET WGSUBS=<path-qualified file name>
//*SET WGRSTRT=<path-qualified file name>
//*****
//*
//* Start WSGrid with JZOS Launcher
//*
//*****
//JAVA EXEC PROC=JVMPRC50,
// JAVACLS='com.ibm.ws.bootstrap.WSLauncher'
//MAINARGS DD *
com.ibm.ws.grid.comm.WSGrid
//*****
//*
//* WSGrid Control Properties - required DD
//*
//* scheduler-host=<host name of job scheduler server>
//* scheduler-port=<HTTP port of job scheduler server>
//* submitter-userid=<authorized userid>
//* submitter-password=<password - may be encoded with WAS utility>
//* timeout=<JMS receive timeout in milliseconds>
//* debug=<true|false>
//*
//*****
//WGCNTL DD PATH='&WGCNTL.'
//*
//*****
//* WSGrid Job Definition - required DD
//*
//* File must contain valid WCG JCL job definition.
//*
//*****
//WGJOB DD PATH='&WGJOB.'
//*
//*****
//* WSGrid Job Substitutions - optional DD
//*
//* Values are of form:
//* substitution-prop.<property name>=<property value>
//*
//*****
//WGSUBS DD PATH='&WGSUBS.'
//*
//*****
//* WSGrid Restart Token - optional DD
//*
//* Note: restart token is written if and only if this job step ends
//*       with RC=4084 (-12).
//*
//*       The WGRSTRT DD takes precedence over the WGJOB DD, so
//*       if this job is restarted with a non-empty restart file
//*       the WCG job identified by the token will be restarted - a
//*       new job instance based on the job definition in WGJOB is
//*       not created.
//*
//*****
//WGRSTRT DD PATH='&WGRSTRT.'
//*           PATHOPTS=(ORDWR,OCREAT),
//*           PATHMODE=(SIRUSR,SIWUSR)
//*
//*****
//* Environment Variable Section
//*
//*****
//STDENV DD *
#

```

```

#-----
# Required: specify WAS home directory
#-----
#
WAS_HOME="<WAS home directory - e.g. /WebSphere/AppServer>"
#
#-----
# Configure JVM options
#-----
#
# Heap size
#
#-----
IJO="-Xms256m -Xmx512m"
#-----
#
# DD encodings
#
# Values:  IBM-1047  (ebcdic - default)
#          IS08859-1 (ascii)
#
#-----
#IJO="$IJO -Dcom.ibm.ws.grid.dd.wgcntl.encoding=IS08859-1"
#IJO="$IJO -Dcom.ibm.ws.grid.dd.wgjob.encoding=IS08859-1"
#IJO="$IJO -Dcom.ibm.ws.grid.dd.wgsubs.encoding=IS08859-1"
#IJO="$IJO -Dcom.ibm.ws.grid.dd.wgrstrt.encoding=IS08859-1"
#-----
#
# Enable verbose:class to debug ClassNotFoundException
#
#-----
#IJO="$IJO -verbose:class"
#-----

REPLACE_WAS_HOME=$WAS_HOME

JAVA_HOME=$WAS_HOME/java
PATH=/bin:/usr/bin:$JAVA_HOME/bin:$WAS_HOME/bin:$PATH
LIBPATH=/lib:/usr/lib:$JAVA_HOME/bin:$JAVA_HOME/bin/classic:$LIBPATH

. $WAS_HOME/bin/setupCmdLine.sh

ENCODE_ARGS="-Xnoargsconversion -Dfile.encoding=IS08859-1"

JMS_PATH=$WAS_HOME/lib/WMQ/java/lib/com.ibm.mq.jar:$WAS_HOME/lib/WMQ/java/lib/co

CLASSPATH=$JAVA_HOME:$WAS_HOME/lib/launchclient.jar:$WAS_CLASSPATH:$JMS_PATH

IBM_JAVA_OPTIONS="$IJO \
-Dwas.install.root=$WAS_HOME \
-Dws.ext.dirs=$WAS_EXT_DIRS \
-Dfile.encoding=IS08859-1 "

export IBM_JAVA_OPTIONS JAVA_HOME PATH LIBPATH CLASSPATH WAS_HOME

//

```

Batch job properties

Use the WSGrid command-line utility to facilitate control of WebSphere batch jobs by external workload schedulers such as the Tivoli Workload Scheduler. You can use these job properties to describe any job type.

Batch control properties:

Properties specified through the WSGrid properties file describe the batch job to be run under the control of WSGrid. This topic describes a set of control properties.

Table 81. Control properties. The table includes a list of property names with a description for each.

Property name	Description
scheduler-connection-factory	The JNDI name of the Java Message Service (JMS) connection factory for the job scheduler message-driver interface application. The default value is <code>jms/com.ibm.ws.grid.ConnectionFactory</code> .
scheduler-host	Specifies the host address of the job scheduler server
scheduler-input-queue	The JNDI name of the JMS input queue for the job scheduler message-driver interface application. The default value is <code>jms/com.ibm.ws.grid.InputQueue</code> .
scheduler-output-queue	The JNDI name of the JMS output queue for the job scheduler message-driver interface application. The default value is <code>jms/com.ibm.ws.grid.OutputQueue</code> .
scheduler-port	Specifies the HTTP port address of job scheduler server.
submitter-userid	Specifies the identity of the job submitter. This ID must be assigned to either the <code>lsubmitter</code> role or <code>lradmin</code> role in order to submit jobs.
submitter-password	Specifies the password of the job submitter. The password might be obfuscated using the WebSphere <code>PropFilePasswordEncoder</code> utility.
debug	Specify <code>true</code> to send debug output to the standard output stream.
timeout	Specifies the timeout value for individual messages sent back from the job scheduler. Units are in milliseconds. The default is 5000.
submit-timeout	Specifies the amount of time in milliseconds the WSGRID utility waits for an initial response from the job scheduler. If this timeout expires, WSGRID concludes the job scheduler is not up and exits with an error message and <code>RC=4084 (-16)</code> .
restart-job	Specifies the job ID of the job to restart, for example, <code>restart-job=postingSample:0001</code> .

Common batch job properties:

Properties specified through the WSGrid properties file describe the batch job to be run under the control of WSGrid. This topic describes a set of common properties.

Common properties

The following properties can be specified for any job type. For example, transactional batch, compute-intensive, or native execution.

Table 82. Common properties. The table lists each property followed by a description.

Property name	Description
job-name	Specifies the name of the job.
job-class	Specifies the requested job class.
application-name	Specifies the name of the required application.
controller-jndi-name	Specifies the JNDI lookup name of the job controller bean. Applicable only to transactional batch and compute-intensive jobs types.
repository-job	Specifies the name of an xJCL definition stored in the job repository. This property is mutually exclusive with both an xJCL file specified on the WSGrid command line and a property-specified job definition.
substitution-prop<prop-name>	Specifies the value of a named substitution property. For example, <code>substitution-prop.interval=10</code> specifies a value of 10 for the substitution property named <code>interval</code> . This property can be specified multiple times, once for each distinctly named substitution property of the target job.
prop<prop-name>	Specifies the value of a named input property that is passed to the job. This property applies to transactional batch and compute intensive job types only. That is, <code>prop.interval=10</code> specifies a value of 10 for the input property named <code>interval</code> . This property can be specified multiple times, once for each distinctly named input property of a given job step.

Transactional batch properties:

Properties specified through the WSGrid properties file describe the batch job to be run under the control of WSGrid. This topic describes a set of transactional batch properties.

The following properties are used to describe a transactional batch job composed of a single step. Using properties provides a methodology for describing a transactional batch job without using xJCL. Specifying these properties is mutually exclusive with specification of an xJCL file on the WSGrid command-line invocation and with use of the `repository-job` property.

Table 83. Transactional batch properties. The table includes a list of property names with a description for each.

Property name	Description
checkpoint-algorithm	Specifies the class name of a checkpoint algorithm implementation. For example, a built-in checkpoint algorithm: <code>com.ibm.wsspi.batch.checkpointalgorithms.timebased</code>
checkpoint-algorithm-prop.<prop-name>	Specifies the value of a named checkpoint algorithm property. For example, <code>checkpoint-algorithm.prop.interval=10</code> specifies a value of 10 for the checkpoint property named <code>interval</code> . This property can be specified multiple times, once for each distinctly named checkpoint algorithm property of the specified checkpoint algorithm.
batch-bean-jndi-name	Specifies the JNDI lookup name of the transactional batch bean. For the POJO programming model, this value is <code>com.ibm.ws.batch.DefaultBatchJobStepBean</code> .
bds.<bds name>	Specifies the class name of a named batch data stream (BDS): <code>bds.input=com.cpv.bds</code> . Input specifies a batch data stream with the logical name of input and the implementation class of <code>com.cpv.bds.Input</code> .
bds-prop.<bds-name>.<prop-name>	Specifies the value of a named BDS property. <code>bds-prop.input.interval=10</code> specifies a value of 10 for the BDS property named <code>interval</code> belonging to the <code>bds</code> named input. This property can be specified multiple times, once for each distinctly named BDS property of a named <code>bds</code> .

Compute-intensive and native-execution properties:

Properties specified through the WSGrid properties file describe the batch job to be run under the control of WSGrid. This topic describes a set of compute-intensive and native-execution properties.

Compute-intensive properties

The following properties describe a compute intensive job composed of a single step, providing an xJCL-less methodology for describing a compute intensive job. Specifying these properties is mutually exclusive with specification of an xJCL file on the WSGrid command-line invocation and with use of the repository-job property.

Table 84. Compute-intensive properties. The table lists each property followed by a description.

Property name	Description
ci-class-name	Specifies the name of the implementation class for the compute intensive batch job step.

Native-execution properties

The following properties describe a native execution job composed of a single step, providing an xJCL-less methodology for describing a native execution job. Specifying these properties is mutually exclusive with specification of an xJCL file on the WSGrid command-line invocation and with use of the repository-job property.

Table 85. Native-execution properties. The table lists each property followed by a description.

Property name	Description
executable	Specifies the command-line string to run in a native process.
arg-line	Specifies the command-line argument string to pass to the native executable.
env-var.<envvar name>	Specifies the value of a name environment variable to set in the native process. This property can be specified multiple times, once for each unique environment variable to set for the specified executable.

Chapter 6. Troubleshooting batch applications

You can troubleshoot batch application issues using such things as messages and logging and tracing.

Adding log and trace settings to the batch environment

The batch environment uses the WebSphere Application Server logging and tracing system.

Log and trace settings

Specify the following settings depending on the component:

Table 86. Settings for logging and tracing. The table includes the component in the description column and the settings for the component.

Description	Setting
Scheduler	com.ibm.ws.batch.*=all com.ibm.ws.grid.* = all com.ibm.ws.gridcontainer.*=all
Endpoints	com.ibm.ws.batch.*=all com.ibm.ws.ci.* = all com.ibm.ws.grid.* = all com.ibm.ws.gridcontainer.*=all

Location of log and trace files

Table 87. Log and trace files. The table includes the component in the description column and the location of the log and trace files for the component in the location column.

Description	Location
Schedulers	<user_install_root>/logs/<server_name>
Endpoints	<user-install-root>/logs/<server_name>

Batch common problems

Occasionally, you might encounter behavior in the batch component that is not expected.

Troubleshooting

Use this section to look for solutions to problems when batch is not working, or not working the way that you expect it to.

Job submission fails due to database failures with the default Apache Derby database

- Check for the successful creation of the LRSCHED database in the <user_install_root>/gridDatabase directory.
- Check the file permissions of the database.
- Derby is only supported on a single scheduler configuration. Use a shared RDBMS for cells configured with more than one scheduler. For example, DB2.

Job submission fails when submitting the job definition file

The following message is returned:

```
Unable to submit the job definition <xJCL file> because the application  
that it runs has not been deployed to an endpoint
```

- Ensure that the application is installed on an endpoint server.
- Ensure the job name or the application name specified in the XJCL matches the name of the application.

Job dispatching slowly when large number of jobs (hundreds or thousands) are submitted

Increase the number of dispatcher threads by setting the MaxConcurrentDispatchers custom property in the job scheduler custom properties panel in the administrative console.

Job execution fails due to database failures with the default Derby database

- Check for the successful creation of the LRSCHEM database in the <user_install_root>/gridDatabase directory
- Check the file permissions of the database.

Database errors during the execution of batch jobs with DB2

- Check for the successful creation of the LRSCHEM database.
- Do not use default the Derby data source JNDI name (jdbc/lree) with DB2. Create a data source for non-default Derby databases.
- Check that the WebSphere variable of GRID_ENDPOINT_DATASOURCE is set to the newly created non-default data source.

Jobs are creating files with the server identity

Set the WebSphere variable RUN_JOBS_UNDER_USER_CREDENTIAL to run jobs under the credential of the submitter. Although jobs can run under the credential of the user on distributed and z/OS operating systems, they work slightly differently. On distributed operating systems, files are created with the identity of the server even if the thread has the credential of the user. On z/OS, the Java thread synchronizes with the operating system thread and files are created with the identity of the user.

Batch applications not working with Java 2 Security

Set the WebSphere variable RUN_JOBS_UNDER_USER_CREDENTIAL to run jobs under the credential of the submitter. Although jobs can run under the credential of the user on distributed and z/OS operating systems, they work slightly differently. On distributed operating systems, files are created with the identity of the server even if the thread has the credential of the user. On z/OS, the Java thread synchronizes with the operating system thread and files are created with the identity of the user.

- Ensure that application security is turned on.
- Grant permissions, SecOwnCredentials, and ContextManager.getServerCredential, in the policy file of the application.

Job log viewing from the Job Management Console fails with the following error: Unable to read the job log.

If administrative security is enabled, ensure that application security is also enabled.

Diagnosing batch problems using job logs

Occasionally, you might encounter behavior in the batch component that is not expected.

CWLRB5586I

Job remains in submitted state with the following message: CWLRB5586I: Job cannot be dispatched at this time. Waiting for an endpoint, a job application, or both to become active.

- Ensure that the application and endpoint servers are running.

CWLRB3112E

Job remains in submitted state with the following message: CWLRB3112E: Job could not be dispatched. Required capability was not found.

- Ensure the required-capability operands are configured correctly.
- If the problem remains, recycle the scheduler and endpoint servers.

BusinessGridStatsCache log file

This log file describes the business batch statistics cache.

Location

This file is in the *log_root/visualization* directory.

Usage notes

node Specifies the node name.

server Specifies the server name.

tcname
Specifies the transaction class name.

appname
Specifies the application name.

j2eemodname
Specifies the Java Platform, Enterprise Edition (Java EE) module name.

version
Specifies the node version.

dtname
Specifies the deployment target name.

scname
Specifies the service policy name.

nodegroup
Specifies the node group name.

cell Specifies the cell name.

updateTime
Specifies the time of the update.

stats num_requested
Specifies the number of jobs which arrive at the runtime environment (endpoint application) for processing.

num_completed
Specifies the number of jobs which run to completion in the runtime environment.

exec_time
Specifies the average time in milliseconds that jobs spend running.

max_concurrency
Specifies the maximum concurrency level that is attained.

num_queued

Specifies the number of jobs that are queued at the scheduler.

num_dispatched

Specifies the number of jobs that are dispatched at the scheduler.

num_failed

Specifies the number of jobs that failed in the runtime environment.

num_errors

Specifies the number of dispatch errors that occurred for jobs.

queue_time

Specifies the average time in milliseconds that a job spent in the queue.

dispatch_time

Specifies the average time in milliseconds that a job spent being dispatched.

dispatch_error_time

Specifies the average time in milliseconds for jobs spent being dispatched when a dispatch error occurred.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

APACHE INFORMATION. This information may include all or portions of information which IBM obtained under the terms and conditions of the Apache License Version 2.0, January 2004. The information may also consist of voluntary contributions made by many individuals to the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org>. You may obtain a copy of the Apache License at <http://www.apache.org/licenses/LICENSE-2.0>.

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

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site (www.ibm.com/legal/copytrade.shtml).

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

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

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

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

Index

A

- AdminTasks
 - grid scheduler 85
- application edition manager 63
- authority
 - scheduler
 - securing the job scheduler application 19, 20, 21, 23, 24
- Autonomic request flow manager
 - APC 11
 - endpoint selection 11
 - tuning 11
 - UseAPCEndpointSelection 11

B

- batch 115
 - POJO 91, 93, 94, 96, 97, 98, 101, 103, 106, 107, 110
- batch application
 - RAD 161
- batch job steps 121

C

- clone 63
- credentials
 - securing the job scheduler application 27

E

- endpoints
 - grid scheduler 25

G

- generic batch step
 - batch 157, 158, 160
- grid scheduler
 - command line interface 30
 - Web services interface 193

I

- installing the long-running application
 - grid scheduler 161

J

- job logs 42

P

- parallel job manager
 - configuring
 - batch job 81
- patterns
 - batch 151, 153, 154, 155
- POJO
 - batch 120
 - programming model 120
- policy
 - batch 159

R

- required-capability 53
- requirements-based job scheduling
 - grid.apps 53
 - job scheduler 53
- rollout 63

S

- scheduler 18, 47
- security 18, 47
- set up the scheduler database
 - scheduler 15, 16

T

- tutorial 70, 115

U

- unit test environment
 - configuring
 - batch job 82

V

- verify
 - grid scheduler 16

W

- work class 63