

WebSphere Lab Jam
Application Infrastructure
WebSphere Compute Grid

Lab Exercises



Catalog Number

Contents

LAB 1	BATCH APPLICATION DEVELOPMENT	5
	1.1 USING THE BATCH SIMULATOR IN ECLIPSE	5
	1.2 DEVELOP A SIMPLE BATCH APPLICATION USING THE BATCH DATA STREAM FRAMEWORK	10
	1.3 UNIT TEST THE READERS, WRITERS AND STEPS USING THE BATCH SIMULATOR	18
	1.4 CREATING AN EAR FILE FOR THE BATCH APPLICATION USING THE BATCH PACKAGER	20
	1.5 INSTALLING THE BATCH APPLICATION.....	22
	1.6 SUBMITTING XJCL AND EXECUTING THE BATCH APPLICATION.....	26
	1.7 TEST CHECKPOINT AND RESTART SCENARIO	28
LAB 2	DEVELOPING AND EXECUTING PARALLEL JOBS	31
	2.1 "PARALLELIZING" AN EXISTING BATCH APPLICATION.....	32
	2.2 DEPLOYING PARALLEL BATCH APPLICATIONS.....	41
	2.3 EXECUTING A PARALLEL JOB	43
LAB 3	USING WSGRID TO INTEGRATE WITH ENTERPRISE SCHEDULERS.....	52
	3.1 SETTING UP INVOCATION ASSETS FOR BATCH JOBS.....	52
	3.2 SUBMITTING JOBS USING WSGRID	55
APPENDIX A.	NOTICES	67
APPENDIX B.	TRADEMARKS AND COPYRIGHTS	69

THIS PAGE INTENTIONALLY LEFT BLANK

Lab 1 Batch Application Development

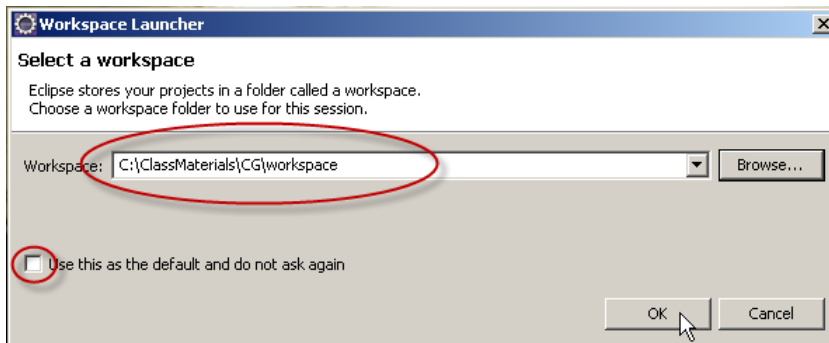
1.1 Using the Batch Simulator in Eclipse

In this lab we cover developing the batch job steps and batch data streams using the batch simulator along in a java perspective in Eclipse. The main point of this section is establishing a workspace containing the batch simulator and then familiarizing ourselves with the contents. In the next section we will develop some batch steps and data streams and unit test them using this workspace.

1. Click on the Eclipse icon on the quick launch bar or find it in the start menu.



2. In the Workspace launcher dialog, enter or browse to **C:\ClassMaterials\CG\workspace** and click **OK**. **Never check** the *use this as the default and do not ask again*. If it takes longer than expected for this dialog to appear, make sure it hasn't been hidden behind the Eclipse splash screen as can happen sometimes. You can use alt-tab to see if it is waiting for input but hidden.



3. The Workspace comes up in the Java™ perspective and contains both an example using the BDS framework and programming directly to the batch container API.

4. The project has an example of a batch application and associated run configurations to test the application. You can verify that the batch simulator is properly configured by executing the tests on these samples. In the Package Explorer open the file **BatchDevEnv-> props.simulator ->Echo.props**. This file is the one that is passed into the batch simulator instead of an xJCL file. It should look like the following:

```
## (C) Copyright IBM Corp. 2008 - All Rights Reserved.
# DISCLAIMER:
# The following source code is sample code created by IBM Corporation.
# This sample code is provided to you solely for the purpose of assisting you
# in the use of the product. The code is provided 'AS IS', without warranty or
# condition of any kind. IBM shall not be liable for any damages arising out of your
# use of the sample code, even if IBM has been advised of the possibility of
# such damages.

job-name=Echo
application-name=Echo

#The following property references the WebSphere XD Compute Grid provided batch controller EJB
#when run in the batch simulator, this actually specifies a pojo wrapper class to the batch step.
#When you deploy this to a batch container running within an application server, this JNDI name
#has to be updated to reference the controller EJB for this step (which is generated for you by
#the batch packager).

controller-jndi-name=ejb/com/ibm/ws/batch/EchoBatchController

#####
# The utilityjars property specifies libraries required by
# this job.
#
# NOTE: this property is used only by the WSBatchPackager utility,
# which is used to create an ear file for deploying this
# batch application.
#

utilityjars=../lib/batchframework.jar;../lib/ibmjzos-1.4.jar

checkpoint-algorithm=com.ibm.wsspi.batch.checkpointalgorithms.RecordbasedBase
checkpoint-algorithm-prop.recordcount=1000

#Input Stream declarations
bds.inputStream=com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteReader
bds-prop.inputStream.PATTERN_IMPL CLASS=com.batch.streams.inputstreams.EchoReader
bds-prop.inputStream.FILENAME=${echo.data}/input.txt
bds-prop.inputStream.debug=false
bds-prop.inputStream.EnablePerformanceMeasurement=false
bds-prop.inputStream.EnableDetailedPerformanceMeasurement=false


#data transformation declarations

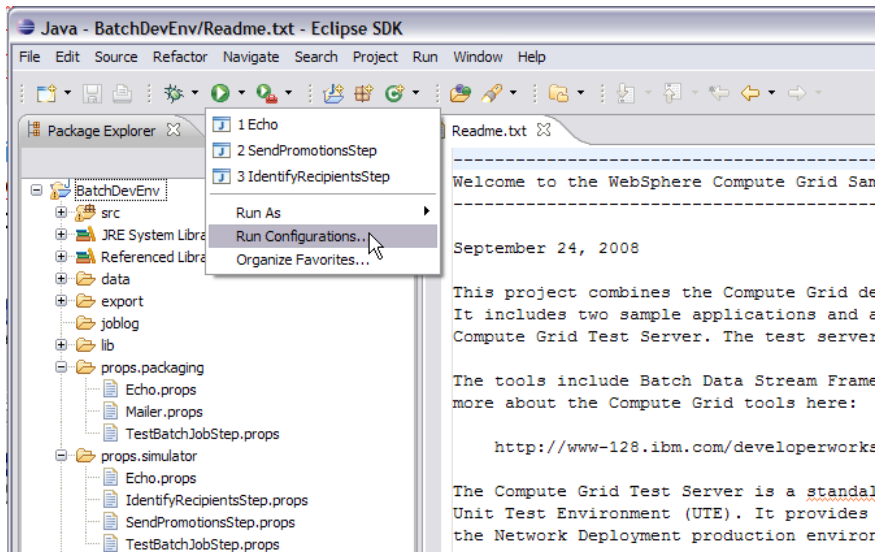
batch_bean-name=IVTStep1
batch-bean-jndi-name=ejb/GenericXDBatchStep
batch-step-class=com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep

#batch-bean-jndi-name=ejb/com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep

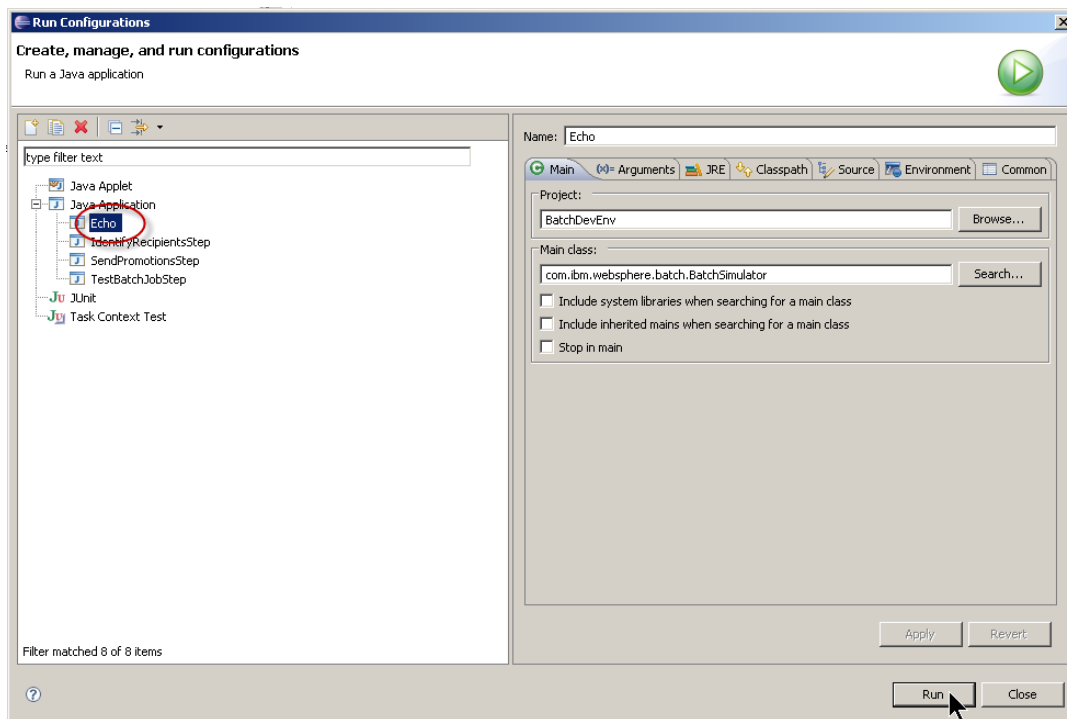
prop.BATCHRECORDPROCESSOR=com.batch.steps.Echo
prop.debug=false
prop.EnablePerformanceMeasurement=false
prop.EnableDetailedPerformanceMeasurement=false

#Output stream declarations
bds.outputStream=com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteWriter
bds-prop.outputStream.PATTERN_IMPL CLASS=com.batch.streams.outputstreams.EchoWriter
bds-prop.outputStream.tablename=alg.tivpwx0
bds-prop.outputStream.FILENAME=${echo.data}/output.txt
bds-prop.outputStream.AppendJobIdToFileName=false
bds-prop.outputStream.EnablePerformanceMeasurement=false
bds-prop.outputStream.EnableDetailedPerformanceMeasurement=false
bds-prop.outputStream.debug=false
```

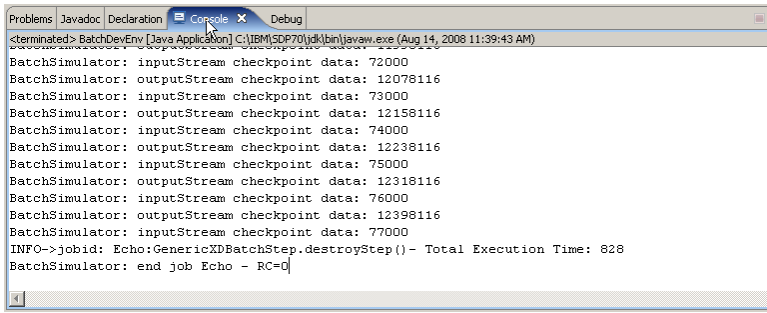
- __5. There has been a run configuration set up in the workspace to easily execute the batch simulator using the Echo.props file shown above. Execute this run configuration by pulling down the menu attached to the Run icon  and selecting **Run Configurations..** as shown below:





- __6. Select **Java Application->Echo** and click **Run**.

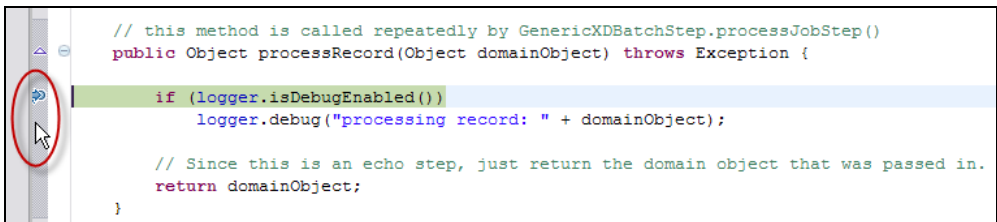


__7. The results will be displayed in the console tab in the lower panel as shown below.

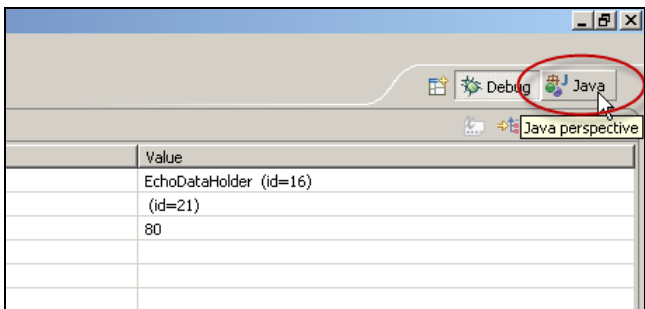


__8. Familiarize yourself with these execution configurations. Also notice that recently used run configurations also appear in the  menu for easy execution.

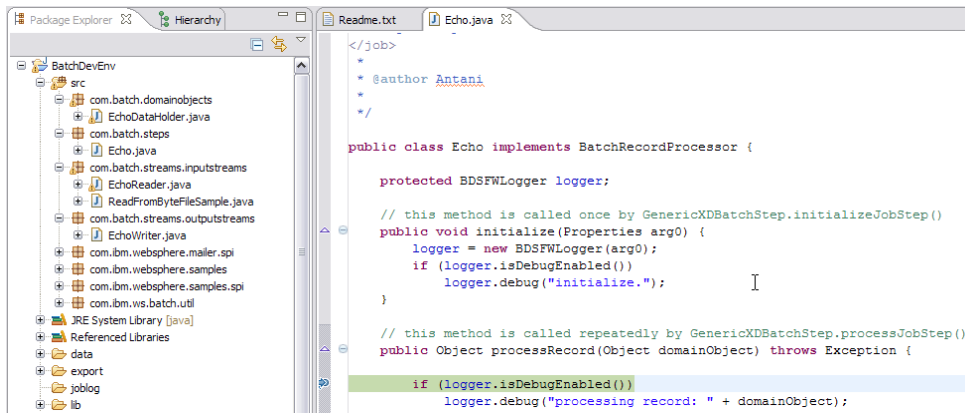
You can also launch the same request in debug mode by pulling down the menu attached to the  debug icon. Try setting some breakpoints in methods in Echo, EchoReader or EchoWriter and then running in debug mode. You can set breakpoints in source code by double-clicking in the very left margin as indicated below. A small blue dot will appear indicating the breakpoint. The green highlighting shown indicates the current execution point during debugging and will not appear initially.



__9. Using the debug functionality will open the debug perspective. To proceed with the rest of this exercise you will want to return to the Java perspective. To do this click on the Java button in the top right corner of Eclipse.



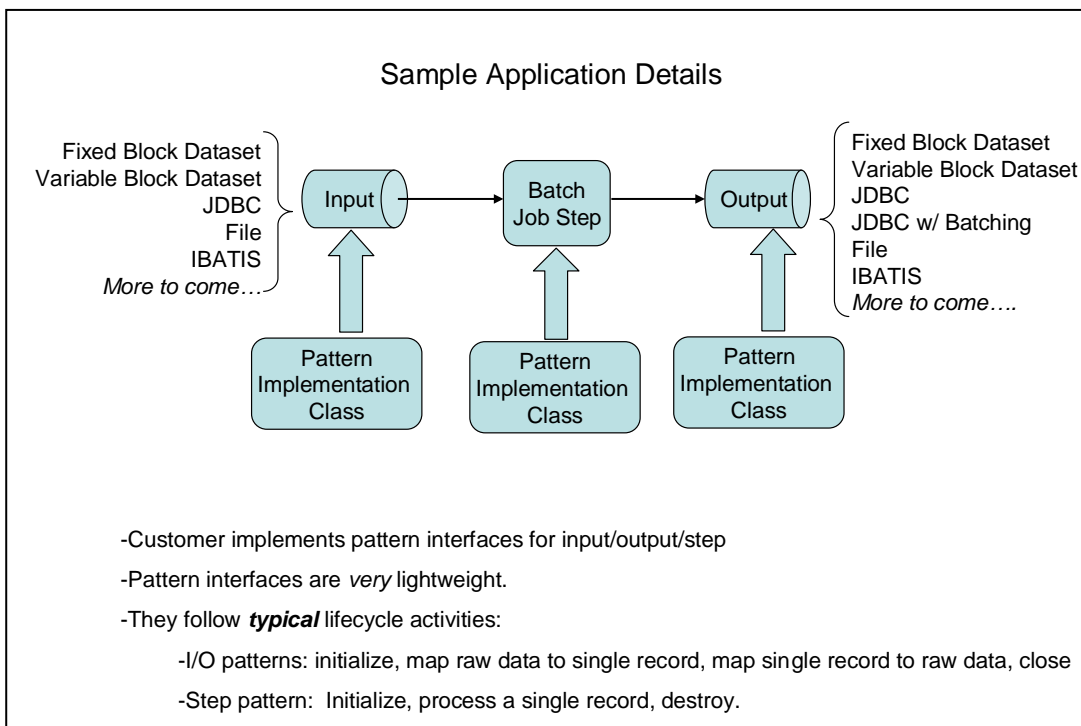
- ___10. Look through the Echo, EchoReader and EchoWriter, This is a simple example of a batch application based on the Batch Data Stream Framework. They can be found by expanding **BatchDevEnv->src** as shown below in packages **com.batch.step.steps**, **com.batch.step.streams.inputstreams** and **com.batch.step.streams.outputstreams**.



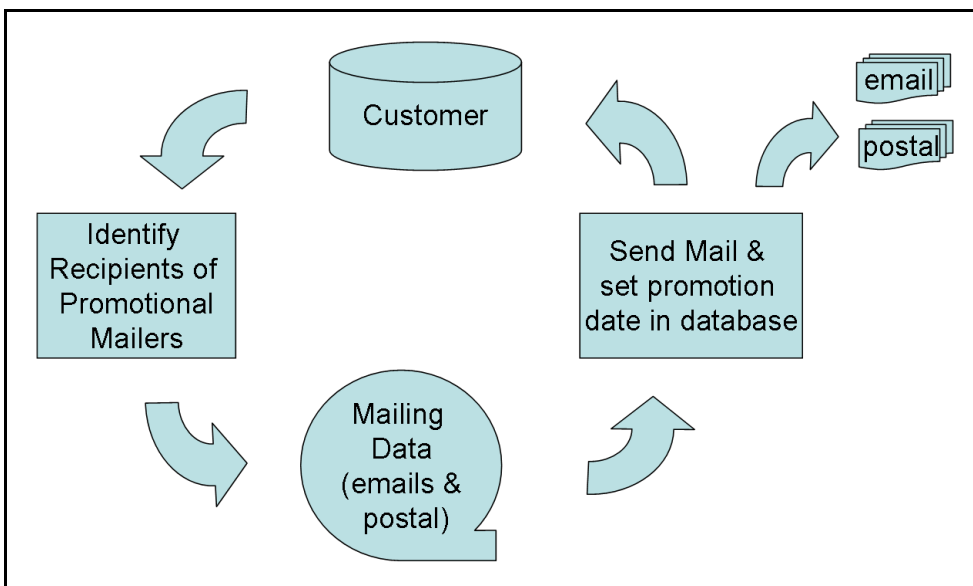
1.2 Develop a simple batch application using the Batch Data Stream Framework

In this section we develop a simple batch application that uses the Batch Data Stream(BDS) Framework. We will develop the Plain Old Java Objects(POJOs) for this application in the batch simulator and do some simple testing.

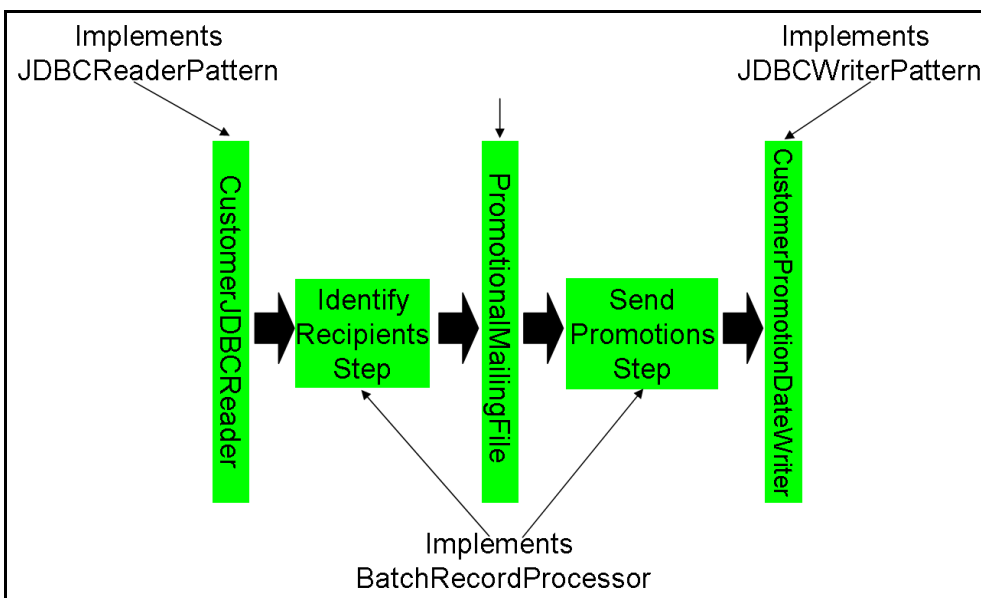
The BDS Framework provides implementations of some commonly occurring data streams as well as providing a pattern based programming model built up around an input-process-output, record-processing metaphor. Without the BDS Framework you were left to implement all data streams and batch steps from scratch. The BDS Framework implements various underlying streams based on files, datasets, JDBC and others, while fulfilling the API contract batch data streams described in the previous chapter. The essential business logic of transforming the underlying stream data to and from meaningful business data is the only part left for you to code. Consider the following diagram that summarizes the approach taken by the BDS framework:



The general flow of the sample application is as follows:

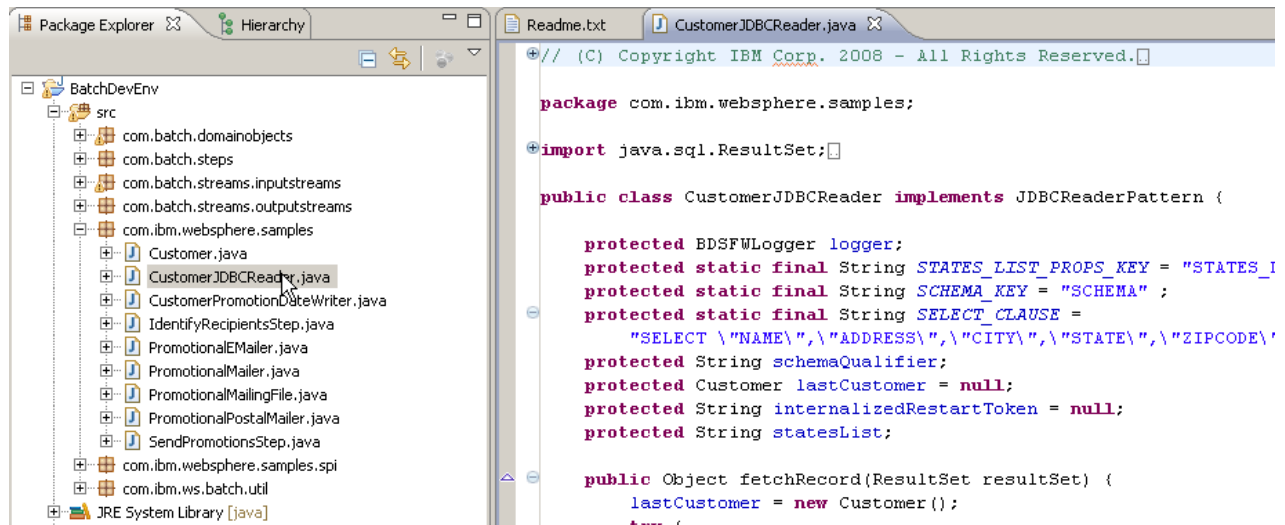


It is implemented as BDS Framework using the following POJOs.



- ___1. For simplicity we will use the batch simulator workspace we used in the previous section. The java files for the Mailer application have already been imported for you. We will go over the important methods of each in the following steps..
- ___2. CustomerJDBCReader implements the JDBCReaderPattern interface. This allows it to be used in an input data stream with any of the associated BSD implementations of JDBCReaders.

In the package explorer, expand **BatchDevEnv->src->com.ibm.websphere.samples** then double-click on **CustomerJDBCReader**.



- ___3. The details of each method will be discussed in each of the following steps in an order that reflects there the order in which they are called by the framework.

- ___4. The initialize method is called when the underlying data stream is initialized and is passed the properties associated with this data stream in the xJCL. Here we will utilize the properties passed in from the xJCL. The constructor for the BDSFWLogger uses the debug property to determine if debug logging is enabled. We pass on the props so it can access this debug property.

The STATES_LIST property is an optional property for CustomerJDBCReader that allows it to be parameterized to only provide customers from certain states. If the STATES_LIST property is set we will parse the list of states and format them in a manner suitable for the query they will be used in later.

The schemaQualifier property has also been added. This allows the various teams in the class to use their own copies of the tables.

```
public void initialize(Properties props) {
    logger = new BDSFWLogger(props);
    schemaQualifier = props.getProperty(SCHEMA_KEY);
    if ( schemaQualifier != null ) {
        schemaQualifier = "\""+schemaQualifier+"\".";
    } else {
        schemaQualifier = "";
    }
    String statesProp = (String)props.get(STATES_LIST_PROPS_KEY);
    if ( statesProp != null ) {
        if (logger.isDebugEnabled()) {
            logger.info("This BDS will only process states: "+statesProp);
        }
        StringTokenizer tok = new StringTokenizer(statesProp, ",");
        statesList = ""+tok.nextToken()+"'";
        while ( tok.hasMoreTokens() ) {
            statesList += ", '"+tok.nextToken()+"'";
        }
    }
}
```

- ___5. The getInitialLookupQuery method is called when the CustomerJDBCReader to get the query that produces the 'stream' of customers desired. This method is called when the job is not running in a restart mode.

```
public String getInitialLookupQuery() {
    String query = SELECT_CLAUSE + schemaQualifier + "\"CUSTOMER\" ";
    if ( statesList != null ) {
        query += " WHERE state in (" + statesList + ") ";
    }
    query += " ORDER BY customerID";
    if (logger.isDebugEnabled() ) {
        logger.info("getInitialPreparedStatement query string:\n\t["+query+"]\n");
    }
    return query;
}
```

- ___6. The `getRestartTokens` method is called when a checkpoint is being taken and returns a String key information required to produce a result set that starts at the same place as the current location in the result set. In this example that data is the `customerID` of the last customer processed.

```
public String getRestartTokens() {
    if ( lastCustomer != null ) {
        return Integer.toString(lastCustomer.getCustomerID());
    } else if ( internalizedRestartToken != null ) {
        return internalizedRestartToken;
    } else {
        return "0";
    }
}
```

- ___7. The `getRestartQuery` differs from `getInitialLookupQuery` in that it is called when the job is restarting and returns a prepared statement for a query that produces the stream of customers starting after the last checkpoint as represented by the data in `restartToken`. Here `restartToken` contains the `customerID` for the last customer processed before the last checkpoint.

```
public String getRestartQuery(String restartToken) {
    String query = SELECT CLAUSE + schemaQualifier + "\"CUSTOMER\" ";
    internalizedRestartToken = restartToken;
    if ( statesList == null ) {
        query += " WHERE customerID > "+restartToken+" ";
    } else {
        query += " WHERE customerID > "+restartToken+" AND state in ("+statesList+" ) ";
    }
    query += " ORDER BY customerID ";
    return query;
}
```

- ___8. The `fetchRecord` method is called repeatedly and returns one instance of customer pulled from the result set each time it is called. The BDS framework does not care what type of object `fetchRecord` returns, but the record processor implementation that will consume these objects should be expecting Customer objects. In this case that is precisely what `IdentifyRecipientsStep` is expecting. .

```
public Object fetchRecord(ResultSet resultSet) {
    lastCustomer = new Customer();
    try {
        lastCustomer.setName(resultSet.getString(1));
        lastCustomer.setAddress(resultSet.getString(2));
        lastCustomer.setCity(resultSet.getString(3));
        lastCustomer.setState(resultSet.getString(4));
        lastCustomer.setZipcode(resultSet.getString(5));
        lastCustomer.setEmail(resultSet.getString(6));
        lastCustomer.setCustomerID(resultSet.getInt(7));
        lastCustomer.setPhone(resultSet.getString(8));
        lastCustomer.setAnnualIncome(resultSet.getInt(9));
        lastCustomer.setLastOfferDate(resultSet.getDate(10));
    }
    catch (Exception e )
    {
        logger.error("Exception in fetchRecord:"+e);
        e.printStackTrace();
        throw new RuntimeException(e);
    }
    return lastCustomer;
}
```

- ___9. Open IdentifyRecipientsStep.java and inspect the code. The BatchRecordProcessor interface that IdentifyRecipientsStep implements has three abstract methods it requires. They are initialize, processRecord and completeProcessing. Note that it is the implementation of processRecord that delivers the business logic that establishes IdentifyRecipientsStep as a simple class that consumes a stream of customers, performs some business logic, in this case comparing their annual income to a marketing threshold, and then produces a corresponding stream of promotional mailers. This is the simple pattern based model shown in the diagram at the beginning of the section.

```
public Object processRecord(Object record) throws Exception {
    Customer cust = (Customer)record;

    PromotionalMailer promo = null;
    if ( cust.getAnnualIncome() > promotionalMailerThreshold ) {
        if ( cust.getEmail() != null && cust.getEmail().length() > 0 ) {
            // setup an e-mail mailer
            promo = new PromotionalEMailer(
                advertisingCampaignCode,
                cust.getCustomerID(),
                cust.getName(),
                cust.getEmail());
        } else if ( cust.postalAddressIsValid() ) {
            // setup an postal mailer
            promo = new PromotionalPostalMailer(
                advertisingCampaignCode,
                cust.getCustomerID(),
                cust.getName(),
                cust.getAddress(),
                cust.getCity(),
                cust.getState(),
                cust.getZipcode());
        } else {
            throw new RuntimeException(
                "Invalid Postal address for customer without email address, CustomerID =" +
                cust.getCustomerID());
        }
    }
    return promo;
}
```

- ___10. A few things to notice about `PromotionalMailingFile`. First, it implements both the `FileWriterPattern` and the `FileReaderPattern`. This was done to keep the code for parsing and formatting records read and written to the corresponding files together. Another thing to notice is that the `fetchRecord` and `writeRecord` methods are implemented specifically to process various derivations of `PromotionalMailer`.

```

public void writeRecord(BufferedWriter out, Object record)
    throws IOException {
    if (record instanceof PromotionalEMailer) {
        PromotionalEMailer promo = (PromotionalEMailer) record;
        out.write(PromotionalEMailer.TYPE_LABEL);
        out.write(TOKEN_DELIMITER);
        out.write(promo.getAdvertisingCampaignCode());
        out.write(TOKEN_DELIMITER);
        out.write(Integer.toString(promo.getCustomerId()));
        out.write(TOKEN_DELIMITER);
        out.write(promo.getCustomerName());
        out.write(TOKEN_DELIMITER);
        out.write(promo.getEmailAddress());
        out.write(LINE_DELIMITER);
    } else if (record instanceof PromotionalPostalMailer) {
        PromotionalPostalMailer promo = (PromotionalPostalMailer) record;
        out.write(PromotionalPostalMailer.TYPE_LABEL);
        out.write(TOKEN_DELIMITER);
        out.write(promo.getAdvertisingCampaignCode());
        out.write(TOKEN_DELIMITER);
        out.write(Integer.toString(promo.getCustomerId()));
        out.write(TOKEN_DELIMITER);
        out.write(promo.getCustomerName());
        out.write(TOKEN_DELIMITER);
        out.write(promo.getAddress());
        out.write(TOKEN_DELIMITER);
        out.write(promo.getCity());
        out.write(TOKEN_DELIMITER);
        out.write(promo.getState());
        out.write(TOKEN_DELIMITER);
        out.write(promo.getZipcode());
        out.write(LINE_DELIMITER);
    } else if ( ! ( record instanceof PromotionalMailer ) ) {
        throw new RuntimeException("SendPromotionsStep.writeRecord presented with" +
            record.getClass().getName()+" expecting sub-class of PromotionalMailer.");
    }
}

public Object fetchRecord(BufferedReader reader) throws IOException {
    PromotionalMailer mailer = null;
    String line = reader.readLine();
    if ( line != null ) {
        StringTokenizer tok = new StringTokenizer(line, TOKEN_DELIMITER);
        String promoType = tok.nextToken();
        if ( promoType.compareTo(PromotionalEMailer.TYPE_LABEL) == 0 ) {
            mailer = new PromotionalEMailer(
                tok.nextToken(),
                Integer.parseInt(tok.nextToken()),
                tok.nextToken(),
                tok.nextToken());
        } else if ( promoType.compareTo(PromotionalPostalMailer.TYPE_LABEL) == 0 ) {
            mailer = new PromotionalPostalMailer(
                tok.nextToken(),
                Integer.parseInt(tok.nextToken()),
                tok.nextToken(),
                tok.nextToken(),
                tok.nextToken(),
                tok.nextToken());
        }
    }
    return mailer;
}

```


- __11. **SendPromotionsStep** step is the simplest class of the five. The business logic in `processRecord` simply takes instances of `PromotionalMailer`, which all must implement a `send` method, calls the `send` method and then returns the `customerID` associated with the promotion just sent.

```
public class SendPromotionsStep implements BatchRecordProcessor {

    public void completeProcessing() {
    }

    public void initialize(Properties props) {
    }

    public Object processRecord(Object record) throws Exception {
        PromotionalMailer promo = (PromotionalMailer)record;
        promo.send();
        return promo.getCustomerId();
    }
}
```

- __12. Finally `CustomerPromotionDateWriter` implements the `JDBCWriterPattern`. It accepts a stream of `customerIDs` and updates the `lastOfferDate` to today's date for the corresponding row in the `CUSTOMER` table.

```
public class CustomerPromotionDateWriter implements JDBCWriterPattern {
    protected final static String QUERY_SUFFIX =
    "\"CUSTOMER\" SET \"LASTOFFERDATE\" = CURRENT_DATE WHERE \"CUSTOMERID\" = ?";
    protected String query_string;
    protected BDSFWLogger logger;
    protected static final String SCHEMA_KEY = "SCHEMA" ;
    protected String schemaQualifier;

    public String getSQLQuery() {
        return query_string;
    }

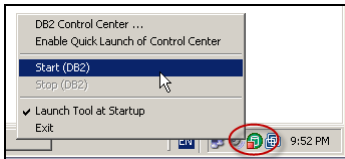
    public void initialize(Properties props) {
        logger = new BDSFWLogger(props);
        schemaQualifier = props.getProperty(SCHEMA_KEY);
        if ( schemaQualifier != null ) {
            query_string = "UPDATE \""+schemaQualifier+"\"."+QUERY_SUFFIX;
        } else {
            query_string = "UPDATE "+QUERY_SUFFIX;
        }
        if ( logger.isDebugEnabled() ) {
            logger.debug("update query string = ["+query_string+"]");
        }
    }

    public PreparedStatement writeRecord(PreparedStatement pstmt, Object record) {
        try {
            pstmt.setInt(1, ((Integer) record).intValue());
        } catch (Exception e) {
            logger.error("Exception in CustomerPromotionDateWriter.writeRecord:" + e);
            e.printStackTrace();
            throw new RuntimeException(e);
        }
        return pstmt;
    }
}
```

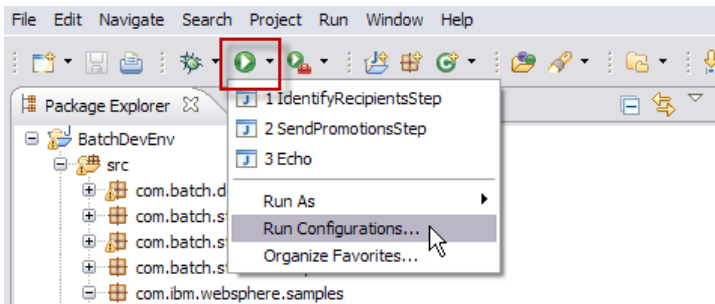
1.3 Unit test the Readers, Writers and Steps using the Batch Simulator

Next, we will use run configurations similar to the examples shown in the first section. These will allow us to drive the batch data steams and job steps through various unit testing scenarios.

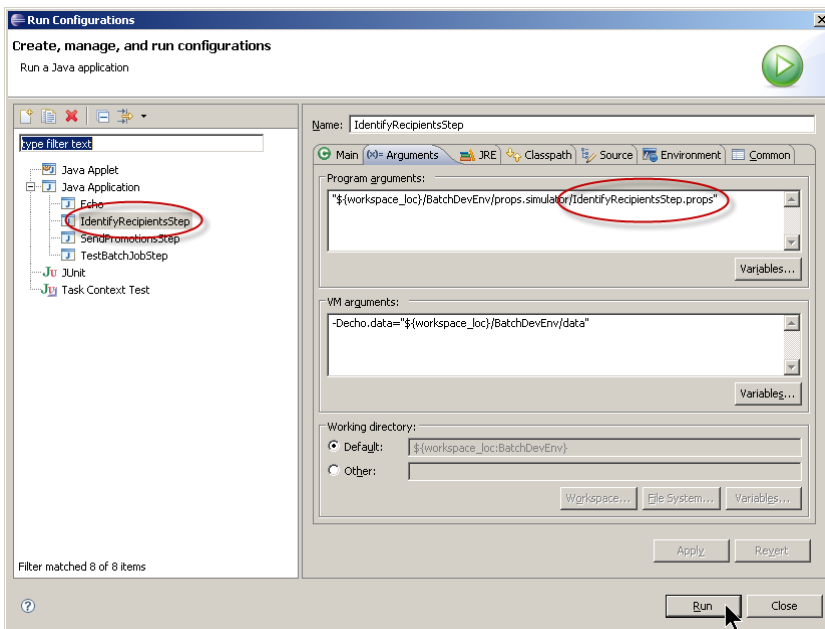
1. Start IBM DB2® by right clicking on the icon in the tray as shown below:



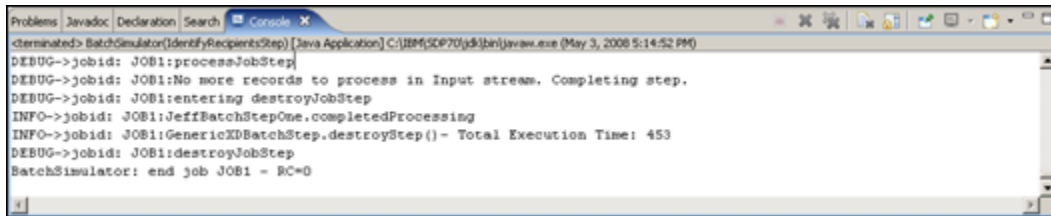
2. Next we will create run configurations to launch the batch simulator referring to these properties files. First select the **BatchDevEnv** project and then from the Eclipse Run menu select **Run Configurations** as shown below.



3. Expand Java Application, select **IdentifyRecipientsStep**. You can click on the **Arguments** tab and see that **IdentifyRecipientsStep.props** is referred to there. This is how the batch simulator is passed the arguments for your streams and steps.



- ___4. After verifying that the changes to the run configuration for **IdentifyRecipientsStep** click **Run**. You should see something like the following in the console window.



```
<terminated> BatchSimulator[IdentifyRecipientsStep] [Java Application] C:\IBM\SCP70\jdk\bin\java.exe (May 3, 2008 5:14:52 PM)
DEBUG->jobid: JOB1:processJobStep
DEBUG->jobid: JOB1:No more records to process in Input stream. Completing step.
DEBUG->jobid: JOB1:entering destroyJobStep
INFO->jobid: JOB1:JeffBatchStepOne.completedProcessing
INFO->jobid: JOB1:GenericXDBatchStep.destroyStep() - Total Execution Time: 453
DEBUG->jobid: JOB1:destroyJobStep
BatchSimulator: end job JOB1 - RC=0
```

- ___5. Open the **IdentifyRecipientsStep.props** by double clicking on it after expanding **props.simulator**. Notice the items defined there and how they relate to properties consumed in the initialize methods of the streams and steps.

You can un-comment the line in section B defining STATES_LIST and re-run the test.

- ___6. Repeat these steps 1 through 4 for **SendPromotionsStep.props** using the run configuration named **SendPromotionsStep**.

1.4 Creating an EAR File for the batch application using the batch packager

The workspace provides ant scripts for various steps in the life cycle of developing the application. In this section we will use one of these scripts to package the application. This will create an Enterprise Application Archive (ER) containing a J2EE™ application that can run inside the Grid Execution Environment(GEE) provided by Compute Grid. .

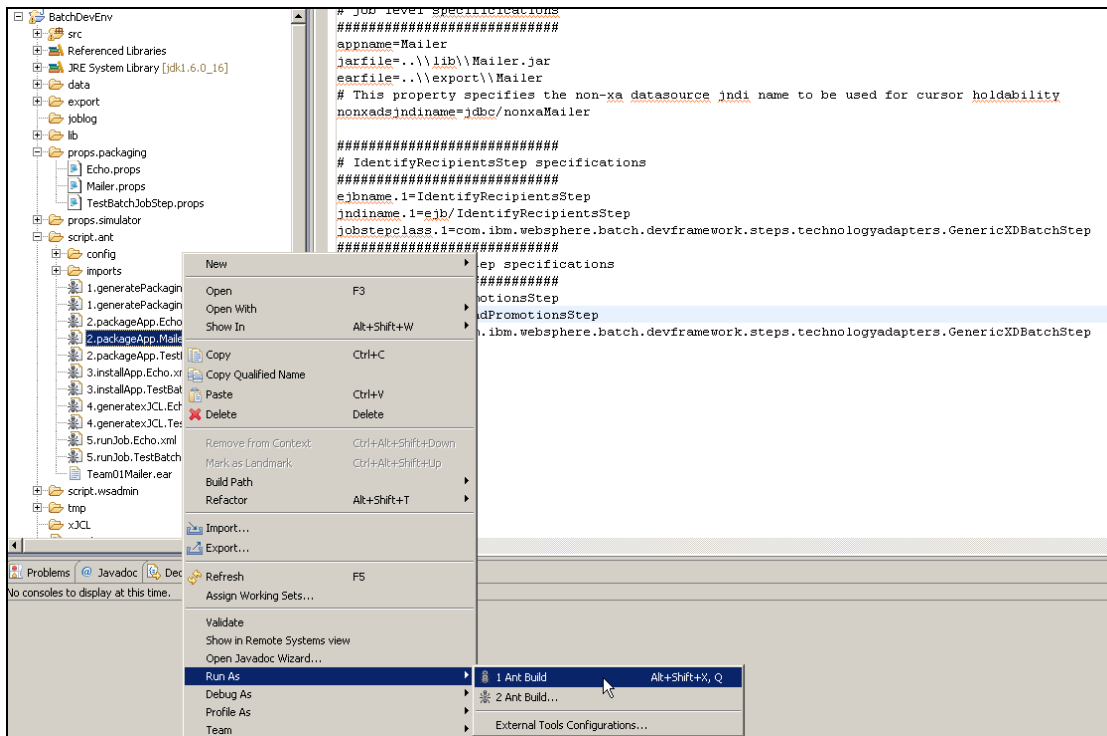
- __1. The batch packager requires various properties for the EAR file and the job steps. These properties can either be passed in as parameters from the command line or can be supplied in a properties file. In the package explorer expand **BatchDevEnv->props.packaging** and double click on **Mailer.props**.
- __2. Inspect this file, it should have the following contents. You will modify this file in the next step.

```
#####
# job level specifications
#####
appname=Mailer
jarfile=..\lib\Mailer.jar
earfile=..\export\Mailer
# This property sets the default JNDI name for the data source used by the Step CMPs to
# access the LREE checkpoint database
epjndiname=jdbc/LREE_DB2

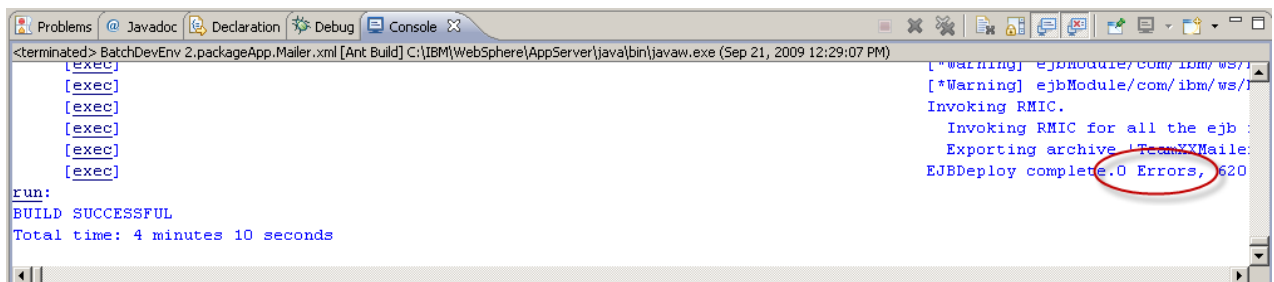
# This property specifies the non-xa datasource jndi name to be used for cursor holdability
nonxadsjndiname=jdbc/nonxaMailer

#####
# IdentifyRecipientsStep specifications
#####
ejbname.1=IdentifyRecipientsStep
jndiname.1=ejb/IdentifyRecipientsStep
jobstepclass.1=com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep
#####
# SendPromotionsStep specifications
#####
ejbname.2=SendPromotionsStep
jndiname.2=ejb/SendPromotionsStep
jobstepclass.2=com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep
```

- ___3. Now execute the ant script to package the application. In the package explorer, right-click on **BatchDevEnv->script.ant->2.packageApp.Mailer.xml** and then select **Run As->Ant Build**.



- ___4. The build will take a few minutes, especially during the execution of the ejbdeploy tool. After a successful execution the Eclipse console view should look like the following. Look for the **0 Errors** message. Don't worry about the warnings. The ejbdeploy tool generates code that produces many warnings in eclipse.



- ___5. Both a pre ejbdeploy EAR and a non-deployed EAR for the application can be found under **C:\ClassMaterials\CG\workspace\BatchDevEnv\export** after the build completes successfully.

1.5 Installing the batch application

In this section we will install your team's batch application using the WebSphere Application Server Integrated Solutions Console (ISC). You might need to start the deployment manager and the node agent.

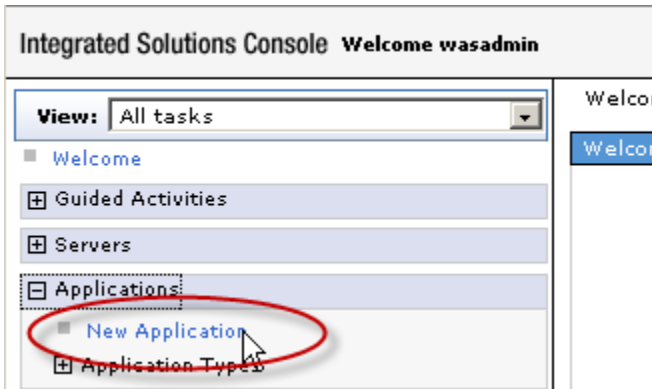
1. There have been shortcuts placed on the desktop to assist you in start them. Click on both in any order and wait for the resulting command prompt windows to close indicating they have finished.



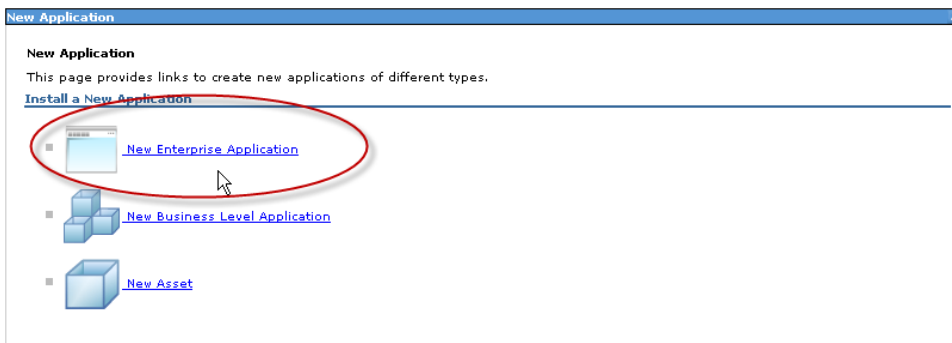
2. The ISC will be used to install the application. There has been a shortcut added to the desktop labeled **ISC**, click on it to open the ISC in a browser window. Enter **wasadmin** for the user id and **wasadmin** for the password.



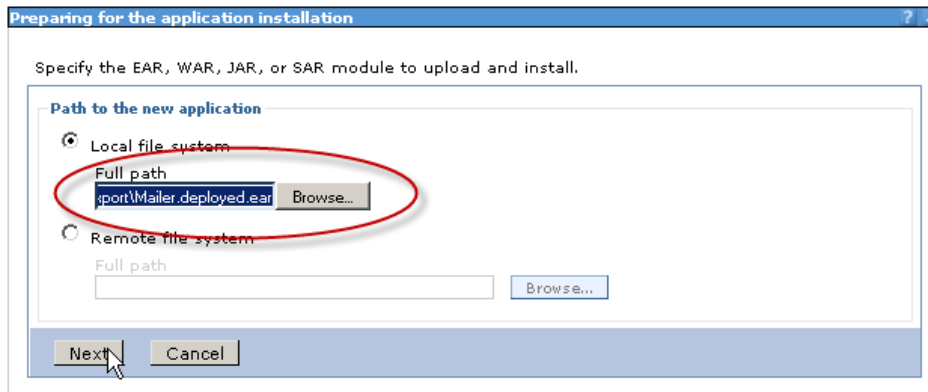
3. In the ISC, Expand Applications and click on **New Application**.



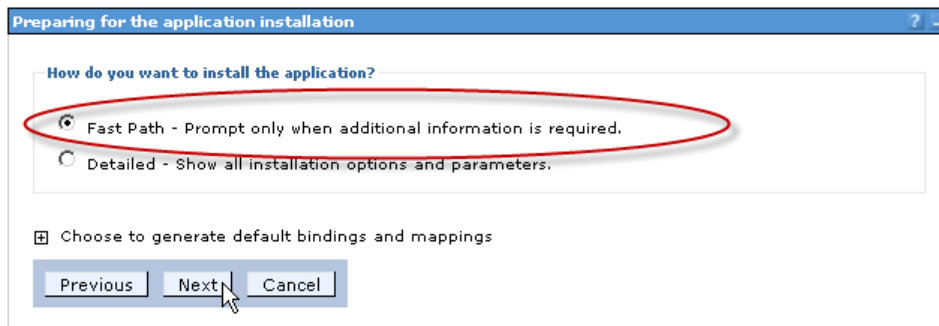
4. Click on **New Enterprise Application**.



- __5. On the next page, select local file system and browse to **C:\ClassMaterials\CG\workspace\BatchDevEnv\export** and select **Mailer.deployed.ear**.

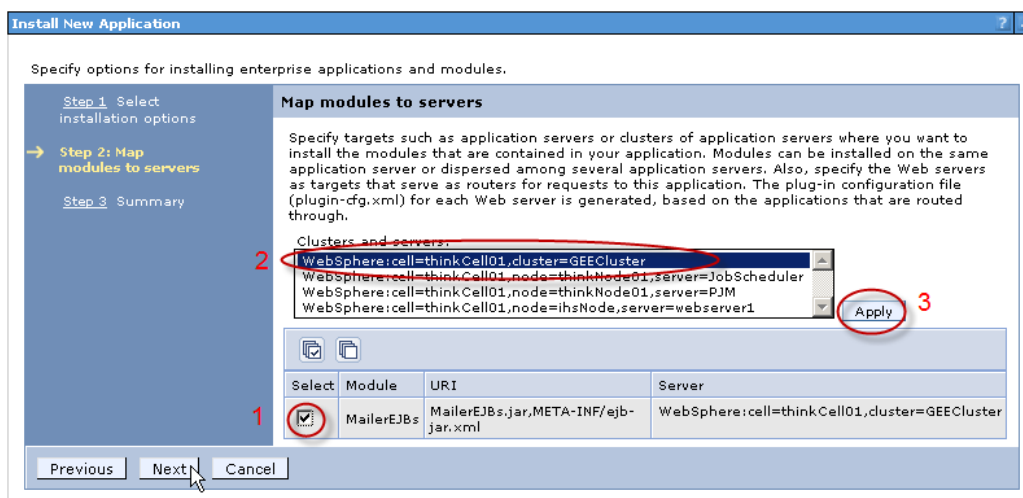


- __6. On the How do you want to install the application page, specify **Fast Path – Prompt only when additional information is required**. Click **Next**.



- __7. On Step 1: Select installation options, accept the defaults. Click **Next**.

- __8. On Step 2: Map modules to servers, check the checkbox next to MailerEJBs and select GEECluster as the target and click **Apply**. Verify that GEECluster is shown in the right most column of the table. Click **Next**.



__9. Inspect the summary page and click **Finish**. The install will start.

Summary

Summary of installation options

Options	Values
Precompile JavaServer Pages files	No
Directory to install application	
Distribute application	Yes
Use Binary Configuration	No
Deploy enterprise beans	No
Application name	Mailer
Create MBeans for resources	Yes
Override class reloading settings for Web and EJB modules	No
Reload interval in seconds	
Deploy Web services	No
Validate Input off/warn/fail	warn
Process embedded configuration	No
File Permission	.*\,dll=755#.*\,so=755#.*\,a=755#.*\,sl=755
Application Build ID	Unknown
Allow dispatching includes to remote resources	No
Allow servicing includes from remote resources	No
Business level application name	
Asynchronous Request Dispatch Type	Disabled
Allow EJB reference targets to resolve automatically	No
Cell/Node/Server	Click here

Warning: No application modules were mapped to Web servers. The plug-in configuration file (plugin-cfg.xml) for each Web server is generated based on the application modules which are mapped to it, therefore no Web server will route requests to this application. To change this option, select the Map modules to servers step.

Previous **Finish** Cancel

__10. When the installation has completed the following will be displayed at the bottom of the screen. Click **Save**. Wait for node synchronization to complete and click **OK**.

ADMA5013: Application Mailer installed successfully.

Application Mailer installed successfully.

To start the application, first save changes to the master configuration.

Changes have been made to your local configuration. You can:

- [Save](#) directly to the master configuration.
- [Review](#) changes before saving or discarding.

To work with installed applications, click the "Manage Applications" link.

[Manage Applications](#)

- __11. You must now start the scheduler server and one of the GEECluster members so both the job scheduler server and the batch application will be available in the next section.

In the ISC, expand **Servers->Server Types** and click on **WebSphere Application Servers**. Check **GEEserver_1** and **JobScheduler** and click **Start**.

The screenshot displays the IBM WebSphere Administration Console interface. On the left, a navigation tree shows 'Servers' expanded to 'Server Types', with 'WebSphere application servers' selected and circled in red. The main content area is titled 'Application servers' and contains a table of server resources. The table has columns for Name, Node, Host Name, Version, Cluster Name, and Status. Two servers are checked: 'GEEserver_1' and 'JobScheduler'. The 'Start' button in the toolbar is also circled in red.

Select	Name	Node	Host Name	Version	Cluster Name	Status
<input checked="" type="checkbox"/>	GEEserver_1	thinkNode01	think.was7.ibm.com	ND 7.0.0.7 WXDCG 6.1.1.0	GEECluster	✘
<input type="checkbox"/>	GEEserver_2	thinkNode01	think.was7.ibm.com	ND 7.0.0.7 WXDCG 6.1.1.0	GEECluster	✘
<input checked="" type="checkbox"/>	JobScheduler	thinkNode01	think.was7.ibm.com	ND 7.0.0.7 WXDCG 6.1.1.0		✘
<input type="checkbox"/>	PJM	thinkNode01	think.was7.ibm.com	ND 7.0.0.7 WXDCG 6.1.1.0		✘

Total 4

1.6 Submitting xJCL and executing the batch application

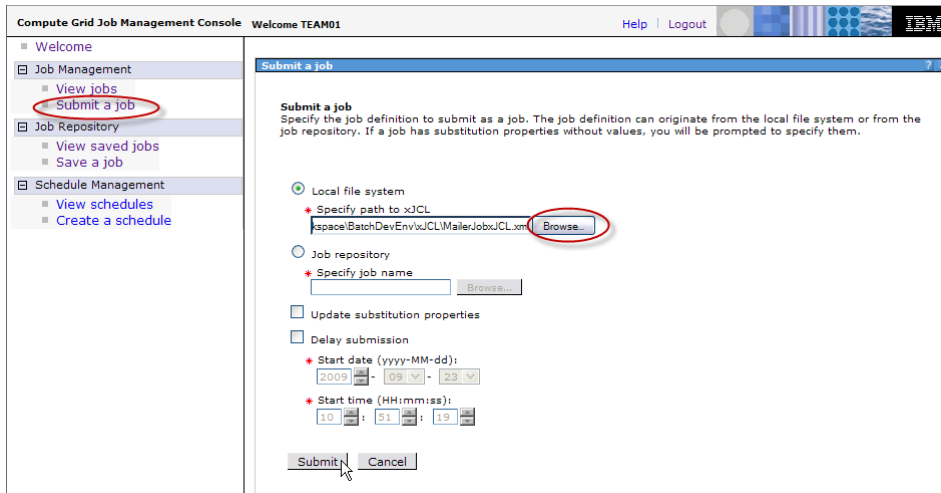
There have been xJCLs provided for the xJCL directory of the workspace at C:\ClassMaterials\CG\workspace\BatchDevEnv\xJCL.

Open the Job Management Console (JMC) by clicking on the shortcut added to the desktop. While the earlier ISC shortcut used Firefox this shortcut uses Internet Explorer. This is important since you will be logging onto the JMC as a different user than you were logging onto the ISC.



Sign on using your **BatchAdmin** and the password (**BatchAdmin**).

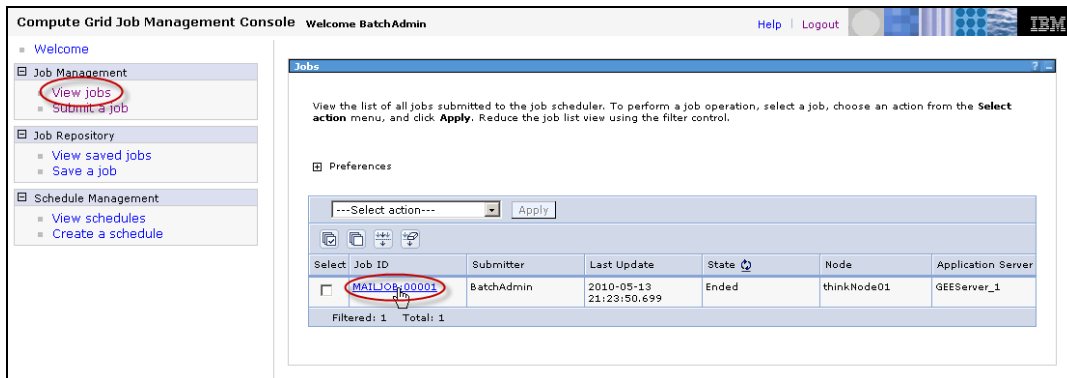
1. Submit the job by browsing to the file at C:\ClassMaterials\CG\workspace\BatchDevEnv\xJCL\MailerJobxJCL.xml
Click **Submit**.



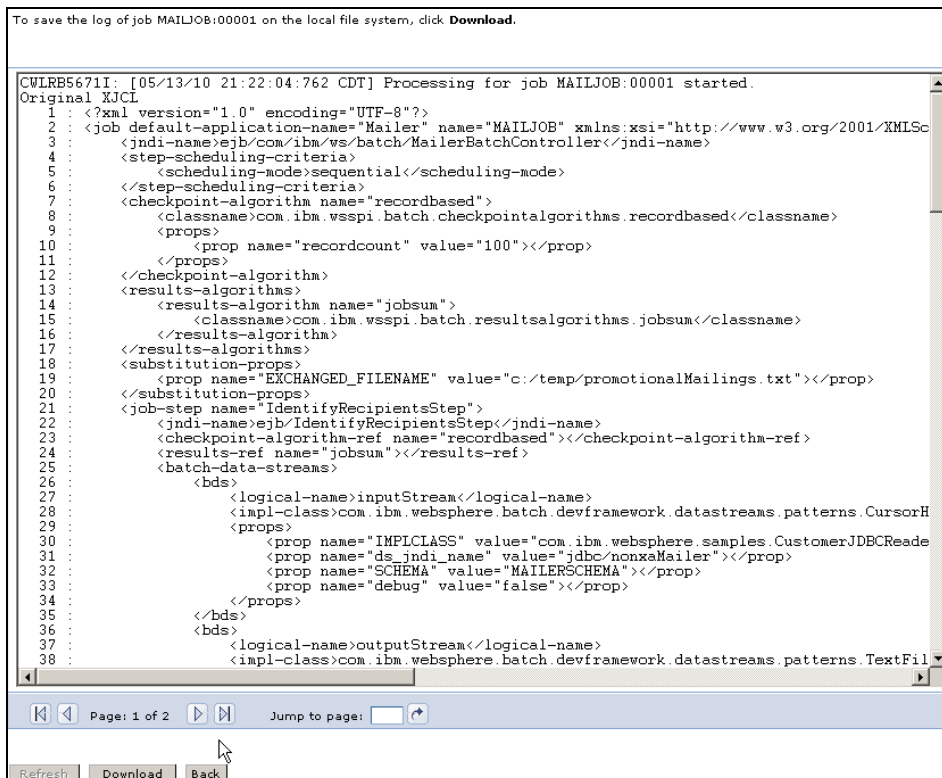
2. When the job is submitted successfully the message shown below should be displayed.



3. You can then view the status of this job along with other jobs by clicking **View jobs**. The job's state should first be submitted and then proceed through executing to ended. You can see the details of the jobs submission by click on the **Job ID**.



4. The log can be viewed in the text area shown or downloaded as a zip file using the button at the bottom. If you scroll through the log you will see the xJCL before and after substitutions, the submission statistics, and the log of the batch application execution along with the final result code. Notice the navigation buttons at the bottom that allow you to navigate multiple pages. The Download button allows you to download a zip file containing the entire log for the job.



1.7 Test Checkpoint and Restart scenario

1.7.1 Execute a batch run that ends in a fails and is restartable

For the purpose of this exercise you will use a Structured Query Language (SQL) script that will modify one of the customer entries in DB2 so that it is invalid. This will cause that customer entry to fail validation logic that is present in the CustomerJDBCReader input stream.

- ___1. Open a DB2 command window by typing **db2cmd** in any of the command prompt windows or in Window's **Start->Run..** dialog. You will use this window in both this and the next section. It will also be useful in other later labs so don't close it when you are done.
- ___2. In the db2 command prompt window, change the current directory using the following command:
cd C:\ClassMaterials\CG\checkpointRestart
- ___3. Enter the following command to run the script that modifies a customer entry to introduce the failure scenario:

db2 -tf injectInvalidCustomerData.sql

```

C:\ClassMaterials\CG\checkpointRestart>db2 -tf injectInvalidCustomerData.sql
SQL1024N  A database connection does not exist.  SQLSTATE=08003

Database Connection Information
Database server      = DB2/NT 9.7.0
SQL authorization ID = DB2ADMIN
Local database alias = MAILER

DB20000I  The SQL command completed successfully.
DB20000I  The SQL command completed successfully.

NAME                                ADDRE
SS                                    CITY
                                     STATE ZIPCODE
-----
EMAIL                                ANNUALINCOME  CUSTOMERID  LASTOFFERDATE
PHONE
-----
Rhona F. Herring                    -
                                     Rochester
                                     TX  41789

-
- 477-774-2159                      18664    1236589  11/04/2009

1 record(s) selected.

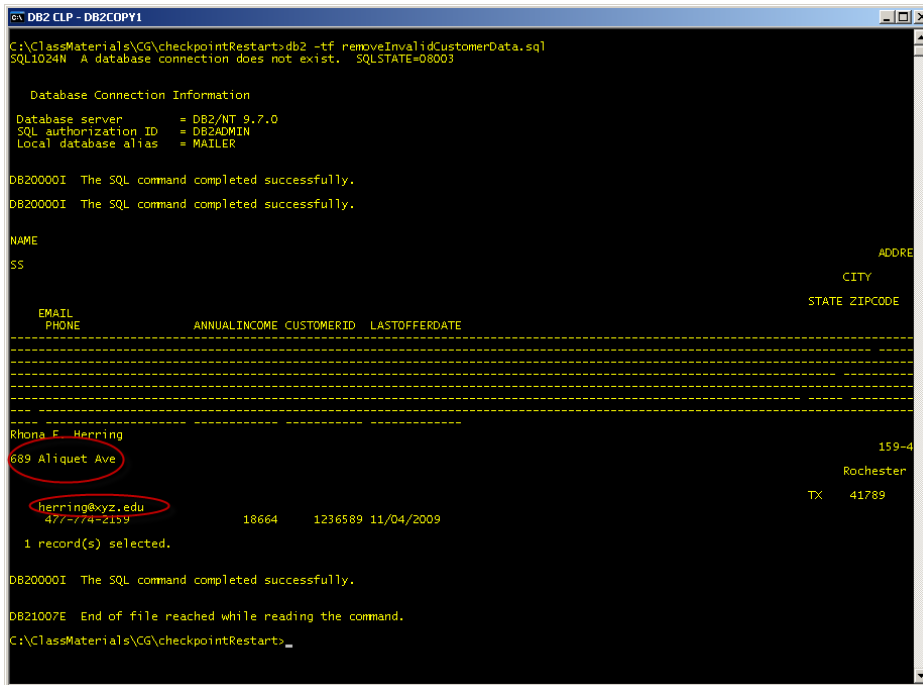
DB20000I  The SQL command completed successfully.

DB21007E  End of file reached while reading the command.
C:\ClassMaterials\CG\checkpointRestart>

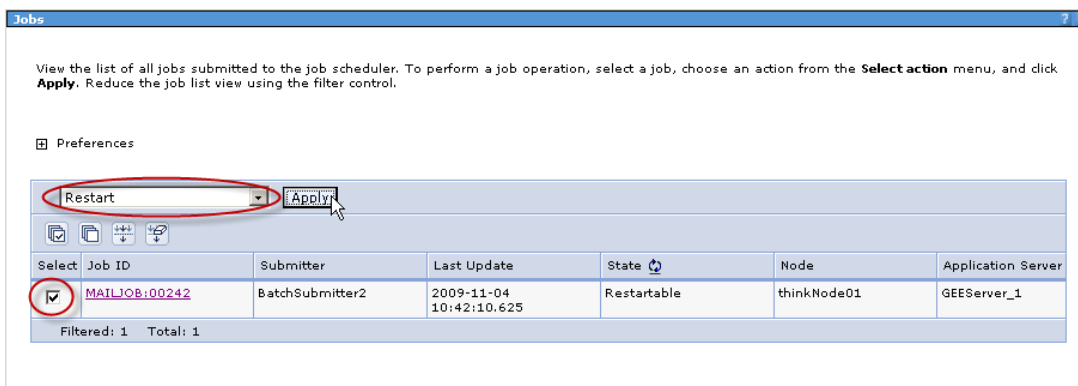
```


1. In the DB2 command prompt window in the same directory as in the previous section, enter the following command. It will modify a customer entry to correct the failure encountered in the previous section.

db2 -tf removeInvalidCustomerData.sql



2. In the JMC, use the check box to select the restartable job from the previous section, select **Restart** from the Select action... pull down menu and then click **Apply**.

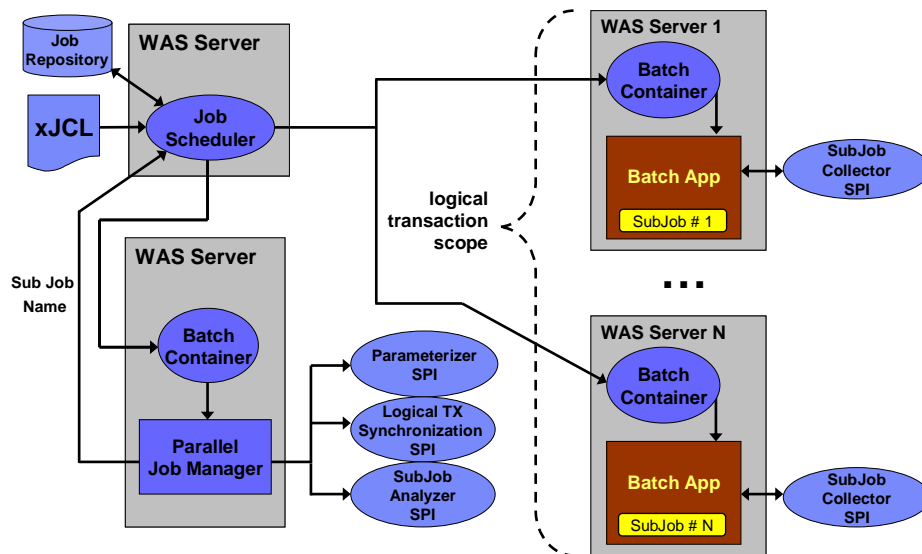


3. The job should reenter the executing state and then eventually reach a state of **ended**. You can inspect the logs for the job. They should contain info from both the initial run and the restart.

Go to the directory C:\temp and compare the size of the file created by this run with the file created by the earlier run.

Lab 2 Developing and executing parallel jobs

The Parallel Job Manager (PJM) is a system provided J2EE application that can be installed in an application server or cluster. It provides the ability to partition one larger batch job into several smaller subjobs and submit them for parallel execution. The PJM manages and monitors the execution of the subjobs as a single cohesive logical job. It uses a concept of a logical transaction to represent the aggregate state of the subjobs and determine the status and completion of the submitted top level job. Provisions are also made for aggregating application specific information during execution from the subjobs returning for analysis in the PJM.



To accomplish this the PJM requires certain information which is acquired via callbacks onto Service Provider Interface(SPI) implementations you provide. The SPIs are as follows:

SPI	
Parameterizer	Called to partition the larger job into subjobs. This partitioning is represented as the number of subjobs and an array of Properties, each of which will be provided to one of the subjobs.
LogicalTX.Synchronization	Called to demarcate the life cycle of a parallel job logical transactions.
SubJobCollector	Gathers application specific information from the executing subjob. An application specified Externalizable object is returned from the callback and propagated to the PJM.
SubJobAnalyzer	Called with the information returned by SubJobCollector and also at each subjob's completion allowing for the aggregation of this information.

2.1 “Parallelizing” an existing batch application

2.1.1 Analyze the batch application.

1. In Eclipse, in the Java perspective, expand **BatchDevEnv->src->com.ibm.websphere.samples** and double click on **CustomerJDBCReader.java** to inspect the code used to read a stream of customers from the database. Find the two methods shown below. They provide the query that retrieves the stream of customers from the database in a regular run and restart run respectively. Notice that the result set can be constrained by providing a list of states from which the customer are to be drawn from. Having such a constraint is a prerequisite for partitioning a large job into multiple subjobs.

```
public String getInitialLookupQuery() {  
  
    String query = SELECT_CLAUSE + schemaQualifier + "\"CUSTOMER\" ";  
    if ( statesList != null ) {  
        query += " WHERE state in (\"+statesList+)\" ";  
    }  
    query += " ORDER BY customerID";  
    if (logger.isDebugEnabled() ) {  
        logger.info("getInitialPreparedStatement query string:\n\t[\"+query+\"]\n");  
    }  
  
    return query;  
  
}  
  
public String getRestartQuery(String restartToken) {  
  
    String query = SELECT_CLAUSE + schemaQualifier + "\"CUSTOMER\" ";  
    int custID = Integer.parseInt(restartToken);  
    if ( statesList == null ) {  
        query += " WHERE customerID > \"+custID+\" ";  
    } else {  
        query += " WHERE customerID > \"+custID+\" AND state in (\"+statesList+)\" ";  
    }  
    query += " ORDER BY customerID ";  
  
    return query;  
  
}
```


- __2. In addition to having a bounding or constraining criteria, these criteria must also be specified in the form of job stream properties. In the same **CustomerJDBCReader.java** file locate the initialize method and observe that it sets `statesList` based on a property passed into the initialize method. These properties are the properties specified in the xJCL as you will verify in the next step.

```
public void initialize(Properties props) {
    logger = new BDSFWLogger(props);
    schemaQualifier = props.getProperty(SCHEMA_KEY);
    if ( schemaQualifier != null ) {
        schemaQualifier = "\"" + schemaQualifier + "\".";
    } else {
        schemaQualifier = "";
    }
    String statesProp = props.getProperty(STATES_LIST_PROPS_KEY);
    if ( statesProp != null ) {
        if (logger.isDebugEnabled()) {
            logger.info("This BDS will only process states: "+statesProp);
        }
        StringTokenizer tok = new StringTokenizer(statesProp, ",");
        statesList = "" + tok.nextToken() + "'";
        while ( tok.hasMoreTokens() ) {
            statesList += ",'" + tok.nextToken() + "'";
        }
    }
}
```

- __3. Now inspect the xJCL for the original job. Expand **BatchDevEnv->xJCL** and double-click on **MailerJobxJCL.xml**. Notice that the properties for the input stream do not mention the `STATES_LIST` property. The xJCL for the parallel job's subjob will need to have this parameter specified correctly.

2.1.2 Define a subjob's xJCL based on the original job's xJCL.

This section describes how a regular job definition like the one discussed above is used to create the definition for the subjobs that will be used by the parallel job manager. The resulting xJCL file has been provided so that you do not need to do the editing yourself.

Typically one starts by copying the original xJCL to a new file and then making two types of modifications. The first modifications are always the same and could be considered 'boiler-plate' modifications. These are a set of substitution and job step properties that you paste into each step definition. They provide the semantics for transmitting certain attributes of the top level job to the subjobs so the relationship can be managed. The other modifications are those you identified in your analysis in the previous section. In this specific case we will be adding a line that includes both a `STATES_LIST` batch data stream property and corresponding substitution property.

- ___4. In eclipse expand **BatchDevEnv->xJCL** and double-click on **MailerSubJobxJCL.xml**. Notice the following change to the first line of the job. This is how the PJM is able to establish the subjobs' names based on the top level job's name.

```
<job name="#{parallel.jobname}" default-application-name="Mailer" xmlns:xsi=...
```

- ___5. Lower in the same file you will find the following block of XML. Each of the job steps requires this to be added. Notice also the last two lines. These are only required if you are using the PJM SPI router to use different SPI implementations for different jobs. All of these are examples of information that the batch container and the framework will need as the subjob executes there.

```
<!-- the following properties are modified at job submission time by the ParallelJobManager. -->
<!-- which are then passed when the sample is submitted from the job repository -->
<!-- these three properties are REQUIRED by ParallelJobManager conventions -->
<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}" />
<!-- This properties is REQUIRED to indicate SubJobCollector implementation in sujob -->
<prop name="PJMRouterAPIs" value="{PJMRouterAPIs}" />
```

- ___6. Finally we see the addition of the STATES_LIST property as shown below. This type of change would not be required if the original xJCL had already expressed this batch data stream property and mapped it to a substitution property.

```
<bds>
<logical-name>inputStream</logical-name>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.CursorHoldableJDBCReader</impl-
<props>
  <prop name="IMPLCLASS" value="com.ibm.websphere.samples.CustomerJDBCReader" />
  <prop name="ds_jndi_name" value="jdbc/nonxaMailer" />
  <prop name="STATES_LIST" value="{STATES_LIST}" />
  <prop name="SCHEMA" value="MAILERSHEMA" />
  <prop name="debug" value="false" />
</props>
</bds>
```

2.1.3 Define the top level job using programmatic subjob partitioning

The partitioning of data to be processed by the various subjobs is accomplished by the Parameterizer. The Parameterizer implements one method called parameterize(). This is passed the step properties for the top level job step and must return two things. First it must return the number of subjobs that the top level job will be partitioned into. It will also return an array of Properties object with each Properties instance in that array being the tailored substitution properties for one of the subjobs.

- ___1. In Eclipse you can see an implementation of that has been provided to partition the Mailer application by state. Expand the **MailerPJMLibrary->src->com.ibm.websphere.mailer.spi** and click on **MailerParameterizer**.

You will notice that this implementation bases the number of subjobs on a step property passed into it. The remainder of the code partitions a list of 50 states into approximately similar sized list of states based on the number of subjobs passed is. These states lists are placed in the various tailored subjob substitution properties along with other step properties that will be passed along to all subjobs untouched.

```
public class MailerParameterizer extends com.ibm.wsspi.batch.parallel.Parameterizer {

protected static final String TEAM_NUM_KEY = "two_digit_team_number" ;
protected static final String STATES_LIST_PROPS_KEY = "STATES_LIST" ;

private static final String states[] =
    {
        "AL","AK","AZ","AR","CA","CO","CT","DE","FL","GA",
        "HI","ID","IL","IN","IA","KS","KY","LA","ME","MD",
        "MA","MI","MN","MS","MO","MT","NE","NV","NH","NJ",
        "NM","NY","NC","ND","OH","OK","OR","PA","RI","SC",
        "SD","TN","TX","UT","VT","VA","WA","WV","WI","WY"
    };

@Override
public Parameters parameterize(String logicalJobName, String LogicalTransactionID, Properties props) {
    System.out.println("com.ibm.websphere.mailer.spi.MailerParameterizer ("
+logicalJobName+", "+LogicalTransactionID+", "+props+") called.");
    // get job count from properties
    int jobcount = Integer.valueOf(props.getProperty("parallel.jobcount","1"));

    Parameters parms = new Parameters();
    parms.setSubJobCount(jobcount);

    //Populate a Properties object for each subjob.
    Properties newprops [] = new Properties[jobcount];
    for ( int i=0; i<jobcount; i++) {
        newprops[i] = new Properties();

        // calculate slice of states array and build
        // state list for subjob
        int slice_size = states.length / jobcount;
        int start_index = i * slice_size;
        int end_index = (i == jobcount-1) ? states.length : (i+1)*slice_size;
        String stateList = new String();
        for ( int j = start_index; j < end_index; j++ ) {
            if ( j == start_index ) {
                stateList += states[j];
            } else {
                stateList += ","+states[j];
            }
        }
        // Assign tailored states list to current subjob
        newprops[i].put(STATES_LIST_PROPS_KEY, stateList);
        // Pass on other properties without tailoring.
        for (Object key : props.keySet() ) {
            newprops[i].put(key, props.get(key));
        }
    }

    parms.setSubJobProperties(newprops);
    return parms;
}
}
```

- __2. The PJM SPI Router allows various application and various top level job to specify which implementation of SPI classes such as the Parameterizer will be used. This is accomplished by packaging SPI implementations in a jar file that contains both the classes and one or more specially named properties files. The name of each properties file represents a logical name for the SPI scheme specified there in. This logical name (excluding the properties extension) is the stated as a step property in the top level job's xJCL.

You can view the properties file that indicates the usage of MailerParameterizer by expanding **MailerPJMLibrary->src** and clicking on **CUSTOM_MAILER.properties**.

```
#
##
### (C) Copyright IBM Corp. 2007 - All Rights Reserved.
### DISCLAIMER:
### The following source code is sample code created by IBM Corporation.
### This sample code is provided to you solely for the purpose of assisting you
### in the use of the product. The code is provided 'AS IS', without warranty or
### condition of any kind. IBM shall not be liable for any damages arising out of your
### use of the sample code, even if IBM has been advised of the possibility of
### such damages.
###
### This file should be copied to ${user.install.root}/properties/xd.spi.properties
##
#

#####
## The following is the SPI implementations for MailerSample application
#####
Parameterizer=com.ibm.websphere.mailer.spi.MailerParameterizer
Synchronization=com.ibm.websphere.mailer.spi.MailerTXSynchronization
SubJobAnalyzer=com.ibm.websphere.mailer.spi.MailerSubJobAnalyzer
SubJobCollector=com.ibm.websphere.mailer.spi.MailerSubJobCollector
```

- __3. Finally a top level job xJCL can be defined that specifies:
- The ParallelJobManager as its target application,
 - All step properties required by the parameterizer logic.
 - A step property **PJMRouterAPIs**, in this case **CUSTOM_MAILER** indicating the SPIs to be used.
 - A step property **com.ibm.wsspi.batch.parallel.subjob.name** with repository name used to save the subjobs xJCL in the next section.

```
<job-step name="Step1">
  <jndi-name>ejb/ParallelJobManager</jndi-name>
  <checkpoint-algorithm-ref name="timebased" />
  <results-ref name="jobsum" />
  <props>
    <prop name="com.ibm.wsspi.batch.parallel.subjob.name" value="MailerSubJob" />
    <prop name="PJMRouterAPIs" value="CUSTOM_MAILER" />
    <!-- The count of parallel subjobs to be submitted -->
    <prop name="parallel.jobcount" value="{parallel.jobcount}" />
    <!-- These properties control the runtime properties generated by the Parameterizer SPI. -->
    <prop name="EXCHANGED_FILENAME" value="c:/temp/PJM-TEST-DATA.txt" />
  </props>
</job-step>
```

2.1.4 Define the top level job using declarative subjob partitioning

There is a BuiltInParameterizer that is provided with the product that allows you to specify the number of subjobs and the tailored subjob substitution properties as a set of step properties in the top level job. The built in parameterizer parses through the step properties and does the following:

- Property `com.ibm.wsspi.batch.parallel.jobs` specifies number of subjobs.
- If the property is of the form `com.ibm.wsspi.batch.parallel.prop.PROPNAME.INDEX` where `PROPNAME` is the substitution property key name and `INDEX` is the sequence number of the subjob (1-N) then subjob `INDEX` will be passed the value indicated.
- All other step properties are passed on unchanged to all subjobs.

1. To use this built in parameterizer, its implementation must be specified as the SPI to be loaded. Using the PJM SPI Router this can be provided in the properties described in the previous section.

You can view the properties file that indicates the usage of the built in parameterizer by expanding **MailerPJMLibrary->src** and clicking on **OUT_OF_BOX.properties**.

```
#
##
### (C) Copyright IBM Corp. 2007 - All Rights Reserved.
### DISCLAIMER:
### The following source code is sample code created by IBM Corporation.
### This sample code is provided to you solely for the purpose of assisting you
### in the use of the product. The code is provided 'AS IS', without warranty or
### condition of any kind. IBM shall not be liable for any damages arising out of your
### use of the sample code, even if IBM has been advised of the possibility of
### such damages.
###
### This file should be copied to ${user.install.root}/properties/xd.spi.properties
##
#

#####
## The following is the spi implementation for MailerSample application
## This set utilizes the built in declarative parameterizer where all
## subjob properties are passed in from the top level job
#####
Parameterizer=com.ibm.ws.batch.parallel.BuiltInParameterizer
Synchronization=com.ibm.websphere.mailer.spi.MailerTXSynchronization
SubJobAnalyzer=com.ibm.websphere.mailer.spi.MailerSubJobAnalyzer
SubJobCollector=com.ibm.websphere.mailer.spi.MailerSubJobCollector
```

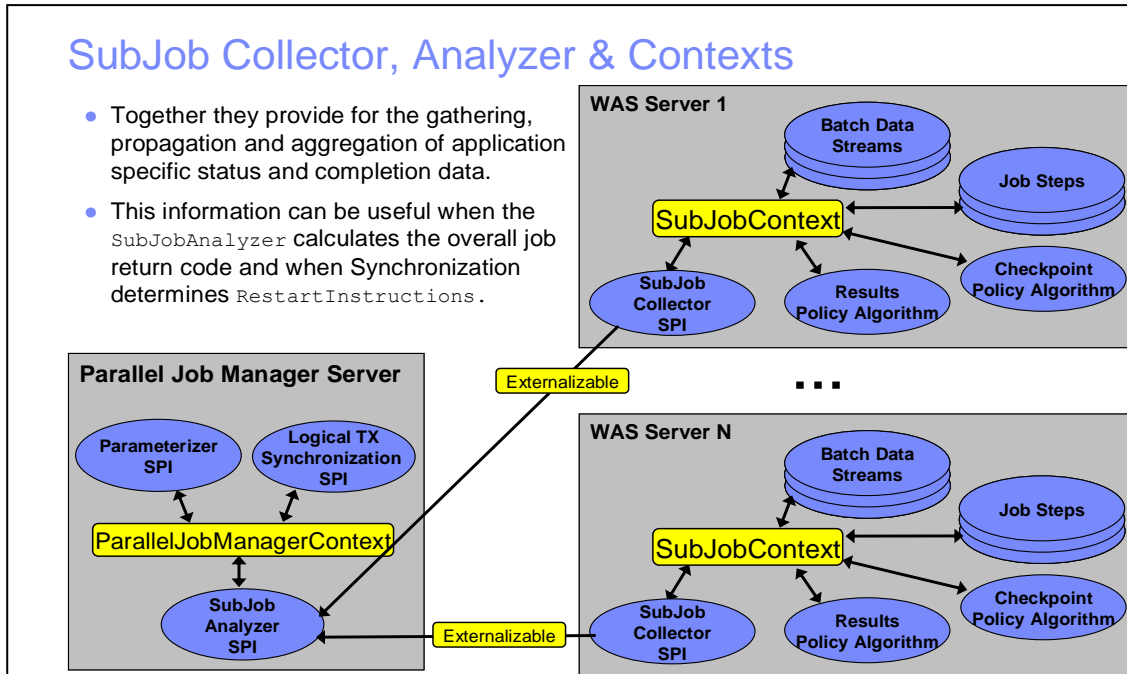
__2. Finally a top level job xJCL can be defined that specifies:

- The ParallelJobManager as its target application
- A step property **PJMRouterAPIs**, in this case **OUT_OF_BOX** indicating the SPIs to be used
- A set of step properties as described in the introduction of this section indicating the number of subjobs and the entire set of tailored subjob substitution properties.
- A step property **com.ibm.wsspi.batch.parallel.subjob.name** with repository name used to save the subjobs xJCL in the next section

```
<job-step name="Step1">
  <jndi-name>ejb/ParallelJobManager</jndi-name>
  <checkpoint-algorithm-ref name="timebased" />
  <results-ref name="jobsum" />
  <props>
    <prop name="com.ibm.wsspi.batch.parallel.subjob.name" value="MailerSubJob" />
    <prop name="PJMRouterAPIs" value="OUT_OF_BOX" />
    <prop name="com.ibm.wsspi.batch.parallel.jobs" value="5" />
    <prop name="EXCHANGED_FILENAME" value="c:/temp/PJM-TEST-DATA.txt" />
    <prop name="com.ibm.wsspi.batch.parallel.prop.STATES_LIST.1"
      value="AL,AK,AZ,AR,CA,CO,CT,DE,FL,GA" />
    <prop name="com.ibm.wsspi.batch.parallel.prop.STATES_LIST.2"
      value="HI,ID,IL,IN,IA,KS,KY,LA,ME,MD" />
    <prop name="com.ibm.wsspi.batch.parallel.prop.STATES_LIST.3"
      value="MA,MI,MN,MS,MO,MT,NE,NV,NH,NJ" />
    <prop name="com.ibm.wsspi.batch.parallel.prop.STATES_LIST.4"
      value="NM,NY,NC,ND,OH,OK,OR,PA,RI,SC" />
    <prop name="com.ibm.wsspi.batch.parallel.prop.STATES_LIST.5"
      value="SD,TN,TX,UT,VT,VA,WA,WV,WI,WY" />
  </props>
</job-step>
```

2.1.5 Modify the application to collect and analyze subjob application statistics

The following diagram illustrates the relationship between the SubJobCollector, the SubJobAnalyzer and the various contexts:



The SubJobCollector returns collected information to the SubJobAnalyzer via an object that implements the `java.io.Externalizable` interface.

1. You can see the implementation of `MailerSubJobCollector` and `MailerSubJobCollector` by expand the **MailerPJMLibrary->src->com.ibm.websphere.mailer.spi** and clicking on the corresponding files.
2. The subjob collector takes the information it gathers off the subjob context. Any of the application code executed by the streams or steps can update the user data carried on this context. The mailer app has two line of code that have been commented out that perform this update.

Examine the code found by expanding **BatchDevEnv->src->cpm.ibm.websphere.samples** and click on **PromotionalEMailer.java**.

```
public void send() {
    CollectedInfoForAnalysis.getCollectedInfo().incrementEmails();
}
```

3. Also notice the same code in the send method of **PromotionalPostalMailer.java**.

- ___4. The code in the two previous steps accumulates counts in an instance of `CollectedInfoForAnalysis` in that is saved away in the `SubJobContext`. Inspect the code for the `SubJobCollector` that gathers that information and returns it to the `Parallel Job Manager`.

Expand **MailerPJMLibrary->src-> com.ibm.websphere.mailer.spi** and click on **MailerSubJobCollector.java**.

```
public Externalizable collect(String logicalJobName, String LogicalTXID, String subJobID) {
    System.out.println(
        "MailerSubJobCollector.collect("+logicalJobName+", "+LogicalTXID+", "+subJobID+"");
    System.out.flush();
    CollectedInfoForAnalysis ret = CollectedInfoForAnalysis.getCollectedInfo();
    ret.markCheckpoint();
    return ret;
}
```

2.1.6 Package the PJM SPI JAR file

The `MailerPJMLibrary` java project in the eclipse workspace represents the Mailer application's SPI jar file. It contains a number of SPI implementations along with some supporting classes. It also contains properties files that identify named usage schemes for these SPIs. There is an ant script provided that packages these into a jar file and places it in the export directory.

- ___1. Expand `MailerPJMLibrary->script.ant`, right-click on `packagePJMLibrary.xml` and select `Run as->Ant build`.

2.2 Deploying parallel batch applications

2.2.1 Install the SPI JAR file

- ___1. Copy the following file to C:\IBM\WebSphere\PJMSharedLibrary
C:\ClassMaterials\CG\workspace\MailerPJMLibrary\export\mailer_spi_library.jar.

This is directory was configured in the PJM server and the GEE cluster servers as a shared library directory.

- ___2. After the jar file has been copied, Restart GEEServer_1 and the start the PJM server. In the ISC, expand **Servers->Server Types** and click on **WebSphere Application Servers**. Check **GEEServer_1** and click **Restart**. After that completes check **PJM** and click **Start**.

Application servers

Use this page to view a list of the application servers in your environment and the status of each of these servers. You can also use this page to change the status of a specific application server.

Preferences

New Delete Templates... Start Stop **Restart** ImmediateStop Terminate

Select	Name	Node	Host Name	Version	Cluster Name	Status
<input checked="" type="checkbox"/>	GEEServer_1	thinkNode01	think.was7.ibm.com	ND 7.0.0.7 WXDCG 6.1.1.0	GEECluster	+
<input type="checkbox"/>	GEEServer_2	thinkNode01	think.was7.ibm.com	ND 7.0.0.7 WXDCG 6.1.1.0	GEECluster	✘
<input type="checkbox"/>	JobScheduler	thinkNode01	think.was7.ibm.com	ND 7.0.0.7 WXDCG 6.1.1.0		+
<input type="checkbox"/>	PJM	thinkNode01	think.was7.ibm.com	ND 7.0.0.7 WXDCG 6.1.1.0		✘

Total 4

Application servers

Messages

thinkNode01/GEEServer_1 server restarted successfully. [View JVM logs](#) for further details.

Use this page to view a list of the application servers in your environment and the status of each of these servers. You can also use this page to change the status of a specific application server.

Preferences

New Delete Templates... **Start** Stop Restart ImmediateStop Terminate

Select	Name	Node	Host Name	Version	Cluster Name	Status
<input type="checkbox"/>	GEEServer_1	thinkNode01	think.was7.ibm.com	ND 7.0.0.7 WXDCG 6.1.1.0	GEECluster	+
<input type="checkbox"/>	GEEServer_2	thinkNode01	think.was7.ibm.com	ND 7.0.0.7 WXDCG 6.1.1.0	GEECluster	✘
<input type="checkbox"/>	JobScheduler	thinkNode01	think.was7.ibm.com	ND 7.0.0.7 WXDCG 6.1.1.0		+
<input checked="" type="checkbox"/>	PJM	thinkNode01	think.was7.ibm.com	ND 7.0.0.7 WXDCG 6.1.1.0		✘

Total 4

2.2.2 Save the subjob xJCL to the job repository.

Since the parallel job manager must submit the various subjobs by name rather than by explicitly providing the xJCL, the subjob definition must be saved to the job schedulers job repository.

- ___1. In the JMC, select **Save a job**. On the resulting page enter **MailerSubJob** as the name and click **Browse...** navigate to **C:\ClassMaterials\CG\workspace\BatchDevEnv\xJCL** and select **MailerSubJobxJCL.xml**. Click **Save**.

Save a job

Specify the job name and the file containing the job definition, then click **Save** to store the job in the job repository.

* Job name:

* xJCL path:

Replace the job if the specified job name exists

- ___2. Save **MailerSubTwoAtOnceJobClass.xml** also located in **C:\ClassMaterials\CG\workspace\BatchDevEnv\xJCL**. This time use the name **MailerSubJobTwoAtOnce**.

Save a job

Specify the job name and the file containing the job definition, then click **Save** to store the job in the job repository.

* Job name:

* xJCL path:

Replace the job if the specified job name exists

2.3 Executing a parallel job

In this section you will test the Mailer app in various parallel processing scenarios. First you will execute a success scenario for both the declarative and programmatic top level job. Next you will use the failure injection technique you used to test check point and restart in the previous exercise to test checkpoint and restart in parallel processing environment. Finally you will see how to tune the submission and dispatching of subjobs.

2.3.1 Test a successful parallel job scenario

__1. From the JMC,

Submit a job

Specify the job definition to submit as a job. The job definition can originate from the local file system or from the job repository. If a job has substitution properties without values, you will be prompted to specify them.

Local file system

* Specify path to xJCL
 DevEnv\xJCL\DeclarativeMailerTopJobxJCL.xml

Job repository

* Specify job name

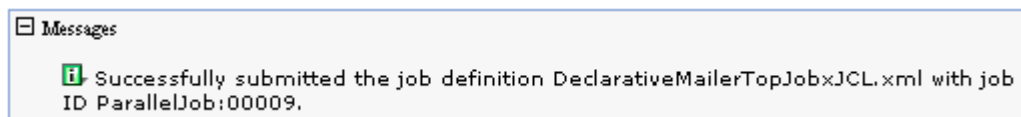
Update substitution properties

Delay submission

* Start date (yyyy-MM-dd):
 2009 - 11 - 06

* Start time (HH:mm:ss):
 05 : 49 : 30

__2. The following message should be displayed indicating that the job has been submitted successfully.



- ___3. To view the status of the submitted top level job, in the JMC, click on **View Jobs**. Notice that both the top level jobs and the subjobs will appear.

Select	Job ID	Submitter	Last Update	State	Node	Application Server
<input type="checkbox"/>	ParallelJob:00009	BatchAdmin	2009-11-06 05:36:23.593	Executing	thinkNode01	PJM
<input type="checkbox"/>	ParallelJob:00009:00010	BatchAdmin	2009-11-06 05:36:47.062	Executing	thinkNode01	GEEServer_1
<input type="checkbox"/>	ParallelJob:00009:00011	BatchAdmin	2009-11-06 05:36:53.281	Executing	thinkNode01	GEEServer_1
<input type="checkbox"/>	ParallelJob:00009:00012	BatchAdmin	2009-11-06 05:36:46.890	Executing	thinkNode01	GEEServer_1
<input type="checkbox"/>	ParallelJob:00009:00013	BatchAdmin	2009-11-06 05:36:47.374	Executing	thinkNode01	GEEServer_1
<input type="checkbox"/>	ParallelJob:00009:00014	BatchAdmin	2009-11-06 05:36:47.093	Executing	thinkNode01	GEEServer_1

Filtered: 6 Total: 6

- ___4. After the top level job completes, click on its job ID and inspect its job log. You will notice that it contains an aggregation of the job logs for the various subjobs.
- ___5. Repeat steps 1 through 4, this time using the **CustomMailerTopJobxJCL.xml**. This will test the use of the MailerParameterizer since the top level xJCL indicates CUSTOM_MAILER as the PJMRouterAPIs value.

2.3.2 Test checkpoint and restart of a parallel job - inject failure

For the purpose of this exercise you will use an SQL script that will modify one of the customer entries in DB2 so that it is invalid. This will cause that customer entry to fail validation logic that is present in the CustomerJDBCReader input stream.

- ___1. Open a DB2 command window by typing **db2cmd** in any of the command prompt windows or in Window's **Start->Run..** dialog. You will use this window in both this and the next section. It will also be useful in other later labs so don't close it when you are done.
- ___2. In the db2 command prompt window, change the current directory using the following command:
cd C:\ClassMaterials\CG\checkpointRestart

- ___3. Enter the following command to run the script that modifies a customer entry to introduce the failure scenario:

db2 -tf injectInvalidCustomerData.sql

```

ex DB2 CLP - DB2COPY1
C:\ClassMaterials\CG\checkpointRestart>db2 -tf injectInvalidCustomerData.sql
SQL1024N  A database connection does not exist.  SQLSTATE=08003

Database Connection Information
Database server      = DB2/NT 9.7.0
SQL authorization ID = DB2ADMIN
Local database alias = MAILER

DB20000I  The SQL command completed successfully.
DB20000I  The SQL command completed successfully.

NAME                ADDRESS
SS                  CITY
                    STATE ZIPCODE
EMAIL
PHONE              ANNUALINCOME CUSTOMERID  LASTOFFERDATE
-----
Rhona F. Herring
                    Rochester
                    TX  41789
-
- 477-774-2159          18664    1236589 11/04/2009
1 record(s) selected.

DB20000I  The SQL command completed successfully.
DB21007E  End of file reached while reading the command.
C:\ClassMaterials\CG\checkpointRestart>
    
```

- ___4. Submit either of the top level jobs from the previous section. from the Job Management Console(JMC) the same manner that you did in the previous section. This time one of the subjobs should end in a restartable state, the top level job should also be in the restartable state.

Select	Job ID	Submitter	Last Update	State	Node	Application Server
<input type="checkbox"/>	ParalleJob:00015	BatchAdmin	2009-11-06 05:52:15.890	Restartable	thinkNode01	PJM
<input type="checkbox"/>	ParalleJob:00015:00016	BatchAdmin	2009-11-06 05:52:09.781	Ended	thinkNode01	GEEServer_1
<input type="checkbox"/>	ParalleJob:00015:00017	BatchAdmin	2009-11-06 05:52:10.328	Ended	thinkNode01	GEEServer_1
<input type="checkbox"/>	ParalleJob:00015:00018	BatchAdmin	2009-11-06 05:52:07.078	Restartable	thinkNode01	GEEServer_1

Filtered: 4 Total: 4

5. In the JMC, on the View Jobs page, double-click on the job id of the subjob you just that failed. Inspect the log for the job. You should see information in the log that indicates the failure that just occurred. You will need to use the navigation buttons at the bottom of the page to go to the last page of the log.

```

Exception: Invalid Postal address for customer without email address, CustomerID = 1236589
ce.samples.IdentifyRecipientsStep.processRecord(Unknown Source)
ce.batch.devframework.steps.technologyadapters.GenericXDBatchStep.processRecord(GenericXDBatchStep.j
ce.batch.devframework.steps.technologyadapters.GenericXDBatchStep.processJobStep(GenericXDBatchStep.j
backLocalException: ; nested exception is: java.lang.RuntimeException: Unexpected error in batch lo
ce.batch.devframework.steps.technologyadapters.GenericXDBatchStep.processJobStep(GenericXDBatchStep.j
DefaultBatchJobStepBean.processJobStep(DefaultBatchJobStepBean.java:87)
ce.batch.EJSLocalCPIIdentifyRecipientsStep_d6a84f9b.processJobStep(Unknown Source)
StepSetup$BatchLoopExecutor.regularBatchJobExecution(StepSetup.java:2425)
StepSetup$BatchLoopExecutor.executeAction(StepSetup.java:2393)
AbstractUserPrivilegedAction.runWithoutSecurity(AbstractUserPrivilegedAction.java:59)
StepSetup.runUnderUserCredential(StepSetup.java:2251)
StepSetup.batchLoop(StepSetup.java:645)
StepSetup.executeStep(StepSetup.java:1120)
JobSetup.callStep(JobSetup.java:1036)
JobSetup.sequentialStepScheduling(JobSetup.java:758)
JobSetup.setupJobSteps(JobSetup.java:557)
ce.batch.BatchEnhancedContainer.DispatchJob(BatchEnhancedContainer.java:45)
BatchJobControllerWork.run(BatchJobControllerWork.java:130)
chbeans.J2EEContext$RunProxy.run(J2EEContext.java:264)
ccessController.doPrivileged(AccessController.java:202)
chbeans.J2EEContext.run(J2EEContext.java:1137)
chbeans.WorkWithExecutionContextIapl.go(WorkWithExecutionContextIapl.java:199)
chbeans.CJWorkItemIapl.run(CJWorkItemIapl.java:188)
d.run(Thread.java:735)
Exception: Unexpected error in batch loop
Exception: Invalid Postal address for customer without email address, CustomerID = 1236589
ce.samples.IdentifyRecipientsStep.processRecord(Unknown Source)
ce.batch.devframework.steps.technologyadapters.GenericXDBatchStep.processRecord(GenericXDBatchStep.j
ce.batch.devframework.steps.technologyadapters.GenericXDBatchStep.processJobStep(GenericXDBatchStep.j

2:10:406 CST] Grid Execution Environment thinkCell101/thinkNode01/GEESEServer 1 caught: javax.ejb.Trans
2:10:500 CST] [Long Running Job Container step execution failed] [Job MAILJOB:00242] [Step IdentifyRe
2:10:500 CST] Destroying job step: IdentifyRecipientsStep
2:10:500 CST] INFO--jobid: MAILJOB:00242:GenericXDBatchStep.destroyStep()- Total Execution Time: 8235
2:10:500 CST] Job step IdentifyRecipientsStep destroy completed with rc: 0
2:10:500 CST] Rolling back step IdentifyRecipientsStep recordbased checkpoint. User transaction stat
2:10:531 CST] Closing IdentifyRecipientsStep batch data stream: inputStream

```

Next you will run another SQL script to repair the customer that you modified in the previous section. Having corrected this problem you will restart the job. Afterward you will observe that the output file for the Identify Recipients job step contains no extra records, the file is the same as the earlier successful run, and that the job completes in an ended state.

- __6. In the DB2 command prompt window in the same directory as in the previous section, enter the following command. It will modify a customer entry to correct the failure encountered in the previous section.

db2 -tf removeInvalidCustomerData.sql

```

C:\ClassMaterials\CG\checkpointRestart>db2 -tf removeInvalidCustomerData.sql
SQL1024N  A database connection does not exist.  SQLSTATE=08003

Database Connection Information
Database server      = DB2/NT 9.7.0
SQL authorization ID = DB2ADMIN
Local database alias = MAILER

DB20000I  The SQL command completed successfully.
DB20000I  The SQL command completed successfully.

NAME                                ADDRE
SS                                  CITY
                                   STATE ZIPCODE
-----
EMAIL                               ANNUALINCOME  CUSTOMERID  LASTOFFERDATE
PHONE
-----
Rhona E. Herring                    159-4
689 Aliquet Ave                    Rochester
herring@xyz.edu                     TX  41789
477-74-2159                          18664  1236589  11/04/2009

1 record(s) selected.

DB20000I  The SQL command completed successfully.

DB21007E  End of file reached while reading the command.
C:\ClassMaterials\CG\checkpointRestart>_

```

- __7. In the JMC, use the check box to select **only the top level job** from the previous section, select **Restart** from the Select action... pull down menu and then click **Apply**.

Select	Job ID	Submitter	Last Update	State	Node	Application Server
<input checked="" type="checkbox"/>	ParallelJob:00015	BatchAdmin	2009-11-06 05:52:15.890	Restartable	thinkNode01	PJM
<input type="checkbox"/>	ParallelJob:00015:00016	BatchAdmin	2009-11-06 05:52:09.781	Ended	thinkNode01	GEEServer_1
<input type="checkbox"/>	ParallelJob:00015:00017	BatchAdmin	2009-11-06 05:52:10.328	Ended	thinkNode01	GEEServer_1
<input type="checkbox"/>	ParallelJob:00015:00018	BatchAdmin	2009-11-06 05:52:07.078	Restartable	thinkNode01	GEEServer_1

Filtered: 4 Total: 4

- 8. The top level job and the failed subjob should noth reenter the executing state and then eventually reach a state of ended.

Select	Job ID	Submitter	Last Update	State	Node	Application Server
<input type="checkbox"/>	ParallelJob:00015	BatchAdmin	2009-11-06 05:58:43.609	Ended	thinkNode01	PJM
<input type="checkbox"/>	ParallelJob:00015:00016	BatchAdmin	2009-11-06 05:52:09.781	Ended	thinkNode01	GEEServer_1
<input type="checkbox"/>	ParallelJob:00015:00017	BatchAdmin	2009-11-06 05:52:10.328	Ended	thinkNode01	GEEServer_1
<input type="checkbox"/>	ParallelJob:00015:00018	BatchAdmin	2009-11-06 05:58:41.031	Ended	thinkNode01	GEEServer_1

Filtered: 4 Total: 4

Go to the directory C:\temp and compare the size of the file created by this run with the file created by the earlier run.

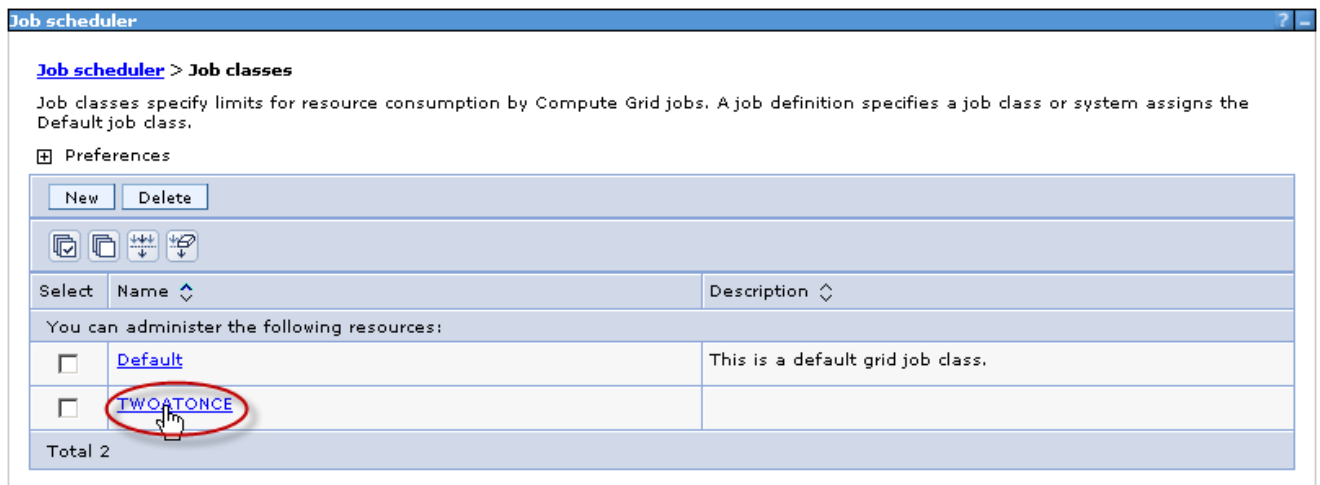
2.3.3 Tuning parallel execution with subjob pacing

In this section you will modify the rate at which subjobs are submitted by the PJM to the job scheduler and you will also modify the rate at which subjobs are dispatched to the endpoints for execution. You will define a special job class for you subjobs to limit the number of concurrently executing jobs of that job class. This job class will be specified in the subjob’s xJCL and will be enforced by the job scheduler. You will also add step properties to the top level job to control the number of concurrently submitted subjobs and the number of threads that will be used for the submissions. These will be enforced by the parallel job manager.

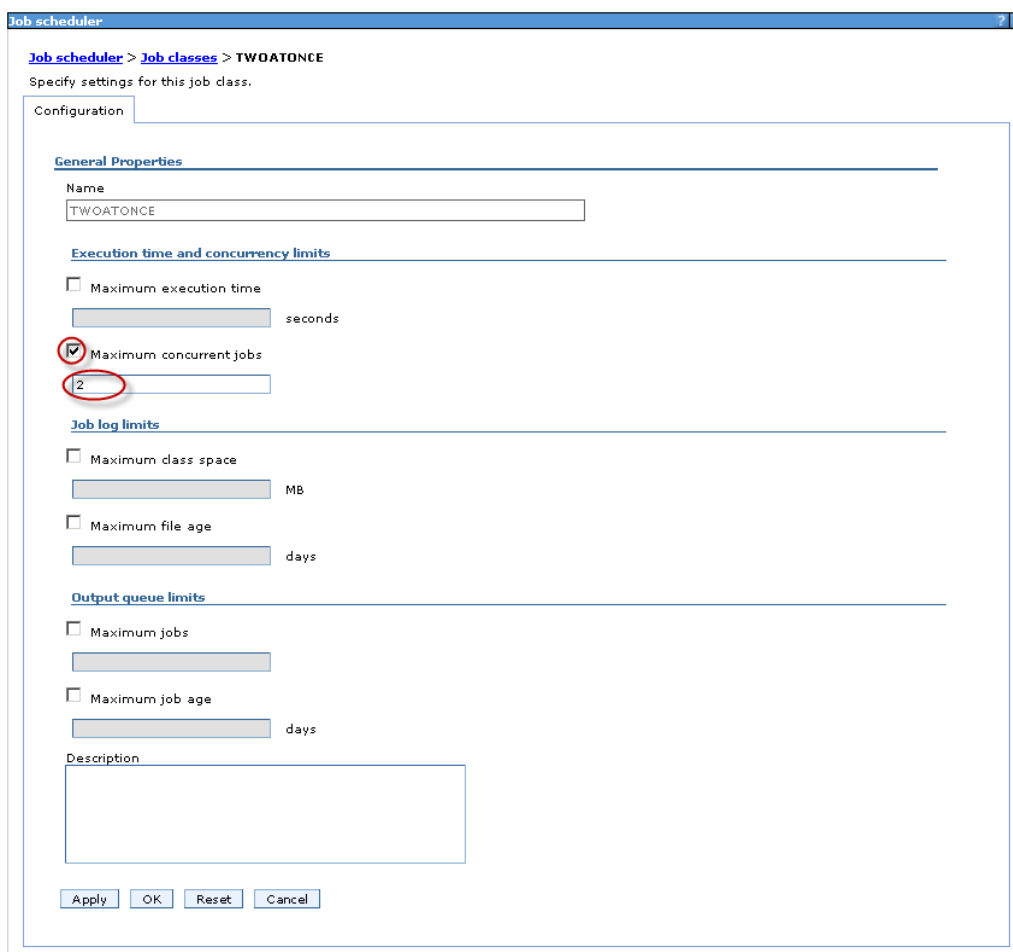
- 1. Inspect the TWOATONCE job class that has been defined to limit the number of jobs that will be dispatched for execution at the same time. In the ISC, expand **System administration** and click on **Job scheduler**. Under the additional properties section click on **Job classes**.

The screenshot shows the 'Job scheduler' configuration window. The 'Additional Properties' section is expanded, showing a list of links: Classification rules, Custom properties, Job classes (circled in red), Security role to user/group mapping, and WebSphere grid endpoints. The 'General Properties' section includes fields for Scheduler hosted by, Database schema name, Data source JNDI name, and Endpoint job log location.

- __2. On the job classes list click **TWOATONCE**.



- __3. On the new job class page, observe that the TWOATONCE job class specifies **2** as the maximum concurrent jobs. Click **Cancel**.



- __4. In Eclipse, expand **BatchDevEnv->xJCL** and open **MailerSubJobTwoAtOnceJCL.xml**. Inspect the xml element defining the job and see that class="TWOATONCE" is specified for the job. This will assign the TWOATONCE job class to the job.

```
<job name="${parallel.jobname}" default-application-name="Mailer" class="TWOATONCE"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

- __5. Save the job to the job repository in the same manner as in the previous section or use the provided script file to do it from the command line. This time use the name **MailerSubJobTwoAtOnce** as the name.

- __6. In Eclipse, expand **BatchDevEnv->xJCL** and open **MailerTopJobWithPacingxJCL.xml**. This is the

```
<job-step name="Step1">
  <jndi-name>ejb/ParallelJobManager</jndi-name>
  <checkpoint-algorithm-ref name="timebased" />
  <results-ref name="jobsum" />
  <props>

    <prop name="com.ibm.wsspi.batch.parallel.subjob.name" value="MailerSubJobTwoAtOnce" />

    <prop name="PJMRouterAPIs" value="CUSTOM_MAILER" />

    <prop name="com.ibm.ws.batch.parallel.MAXIMUM_CONCURRENT_SUBJOBS" value="4" />
    <prop name="com.ibm.ws.batch.parallel.MAXIMUM_SUBJOBS_SUBMISSION_THREADS" value="2" />

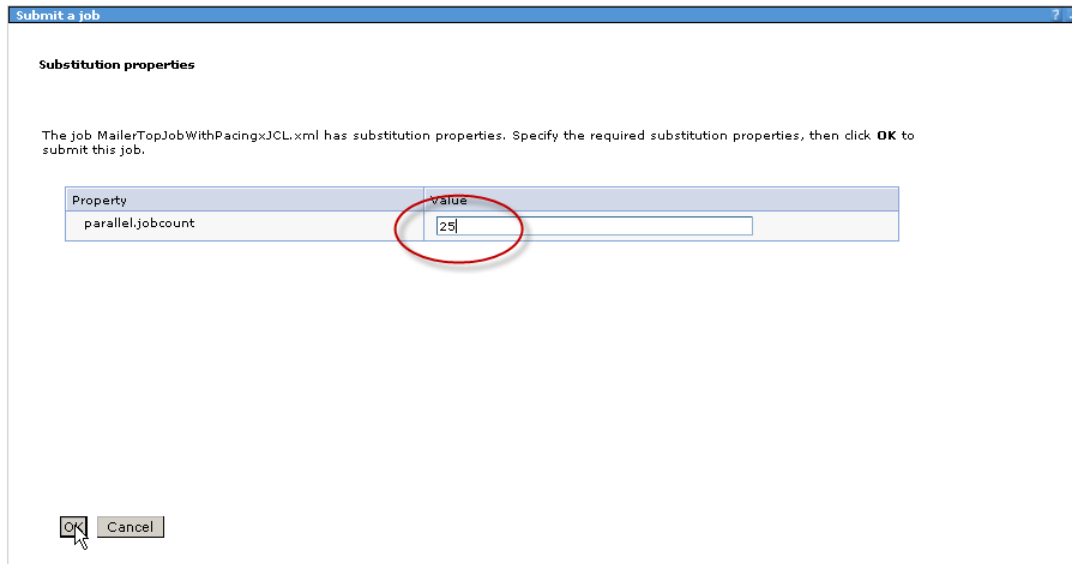
  <!-- The count of parallel subjobs to be submitted -->
  <prop name="parallel.jobcount" value="${parallel.jobcount}" />

  <!-- These properties control the runtime properties generated by the Parameterizer SPI. -->
  <prop name="EXCHANGED_FILENAME" value="c:/temp/PJM-TEST-DATA.txt" />

</props>
</job-step>
```

- __7. Submit the top level job **MailerTopJobWithPacingxJCL.xml** but this time check **Update substitution properties**. Click **Submit**.

__8. Enter **25** for the number of subjobs. Click **OK**.



The screenshot shows a dialog box titled "Submit a job" with a subtitle "Substitution properties". The text inside reads: "The job MailerTopJobWithPacingxJCL.xml has substitution properties. Specify the required substitution properties, then click **OK** to submit this job." Below this text is a table with two columns: "Property" and "Value". The table contains one row with the property "parallel.jobcount" and the value "25". The value "25" is circled in red. At the bottom left of the dialog box, there are two buttons: "OK" and "Cancel".

Property	Value
parallel.jobcount	25

Observe that this time when the subjobs will be submitted four at a time and they will only execute two at a time. You will need to click the refresh icon at the top of the status column to see the changes.

Lab 3 Using WSGrid to integrate with enterprise schedulers

3.1 Setting up invocation assets for batch jobs

In this section you will prepare artifacts that will be used to control the way WSGrid submits jobs. The first of these is the control properties which determine how WSGrid connects to the WebSphere Compute Grid job scheduler. The next file provides job properties. This file provides any substitution properties that are required by the job along with the name of the xJCL if it is being submitted from the job scheduler's job repository.

3.1.1 Define the control properties file

___1. Open a command prompt window and go to the directory:

C:\ClassMaterials\CG\WSGrid\working

___2. Open the file **control.properties** using notepad or notepad++ and observe the contents. The control file contains connection properties and general execution properties for WSGrid. One control file could be used in the submission of various different job types against different batch applications. There would need to be a different control file for a different Compute Grid instance on a different cell. Access to the information in these files is controlled by file system access control.

```
# host of my job scheduler
scheduler-host=think.was7.ibm.com
# http port of my job scheduler server
scheduler-port=9080
# user id of job submitter
submitter-userid=BatchSubmitter1
# job submitter password
submitter-password={xor}HT4rPDcMKj0yNisrOilu

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

___3. Close the file and keep the command prompt window open.

3.1.2 Define the job properties file

- __1. Open the job.properties and observe the properties present there. The property repository-job is the name by which the xJCL for the job has been stored in the Job Scheduler's repository. All properties that begin with the prefix substitution-prop are understood to be substitution properties that will be resolved inside the xJCL. In this example, DEBUG is the only substitution property being passed into the submission.

```
repository-job=NormalWSGridJob
substitution-prop.DEBUG=false
```

- __2. Open the pjmljob.properties and observe the properties present there. This is another example of a job properties file in this case targeting a different repository xJCL and passes in an additional substitution property. Using this job properties file submits the top level job of a parallel job having the indicated number of subjobs.

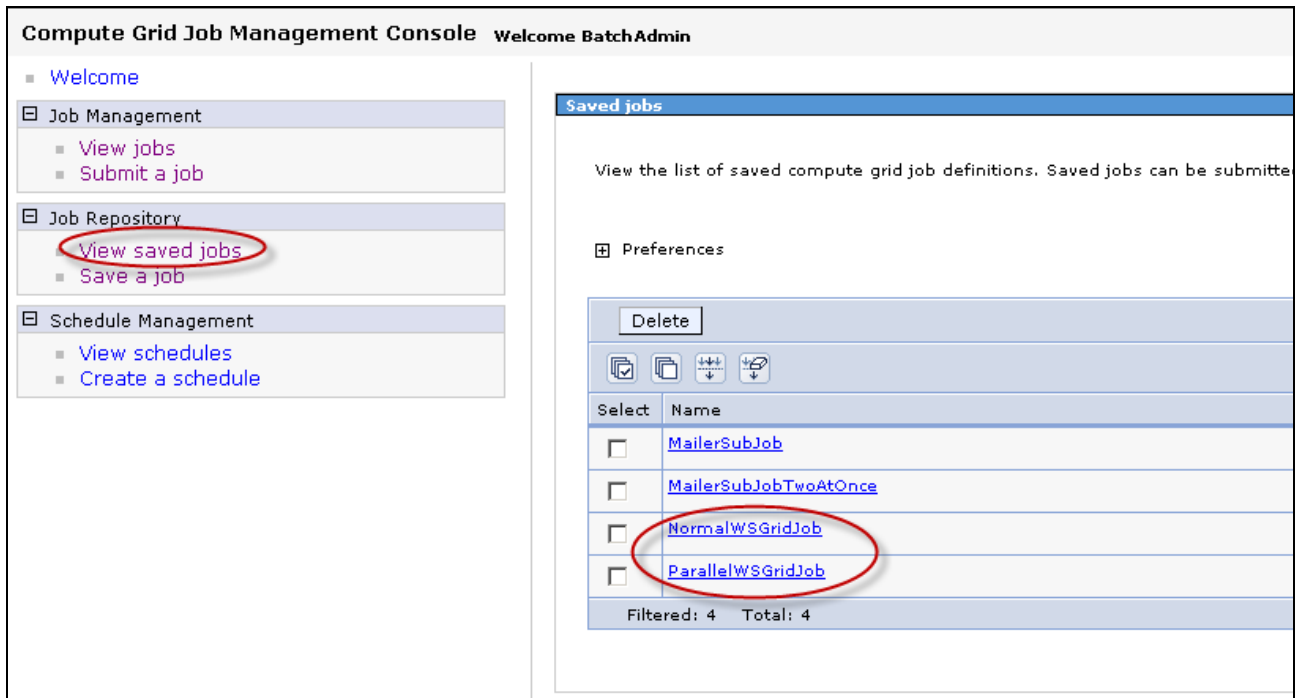
```
repository-job=ParallelWSGridJob
substitution-prop.parallel.jobcount=4
substitution-prop.DEBUG=false
```

- __3. Close both notepad instances.

3.1.3 Store the xJCL in the job repository

The two xJCL files for this exercise have already been saved to the job scheduler's repository.

1. Open the Job Management Console expand **Job Repository** and click on **View saved jobs**. In the list you will see the two names mentioned in the job properties files in the previous section. You can click on each name to view its xJCL. You will recognize them as MailerJobxJCL.xml and CustomMailerTopJobxJCL.xml. These will be submitted in the remaining sections.



The screenshot shows the Compute Grid Job Management Console interface. On the left, the 'Job Repository' section is expanded, and 'View saved jobs' is highlighted with a red circle. The main panel, titled 'Saved jobs', displays a table of job definitions. The table has two columns: 'Select' and 'Name'. The rows are:

Select	Name
<input type="checkbox"/>	MailerSubJob
<input type="checkbox"/>	MailerSubJobTwoAtOnce
<input type="checkbox"/>	NormalWSGridJob
<input type="checkbox"/>	ParallelWSGridJob

At the bottom of the table, it says 'Filtered: 4 Total: 4'. The 'NormalWSGridJob' and 'ParallelWSGridJob' entries are circled in red.

2. You can return to the View jobs page of the JMC after you inspect the two xJCL files. We will use the JMC in the next section to observe the progress of the jobs submitted by WSGrid from a second perspective.

3.2 Submitting jobs using WSGrid

3.2.1 Submit a job from the job repository.

- __1. Return to the command prompt window and the directory
C:\ClassMaterials\CGWSGrid\working
- __2. Launch WSGrid using the control and properties file for the normal job using the following command:

c:\IBM\WebSphere\AppServer\bin\WSGrid.bat control.properties job.properties

The following will be displayed as the job is submitted

```
WSGrid Version UNKNOWN [cf21006.51041] 2010-02-02 00:03:43
CONTROL: scheduler-port=9080
CONTROL: debug=false
CONTROL: submitter-password={xor}HT4rPDcMKj0yNisrOilu
CONTROL: submitter-userid=BatchSubmitter1
CONTROL: scheduler-host=think.was7.ibm.com
CONTROL: timeout=8000
JOB: repository-job=NormalWSGridJob
JOB: substitution-prop.DEBUG=false
May 14, 2010 2:09:46 PM null null
WARNING: ssl.default.password.in.use.CWPKI0041W
May 14, 2010 2:09:46 PM null null
INFO: ssl.disable.url.hostname.verification.CWPKI0027I
May 14, 2010 2:09:47 PM null null
INFO: Client code attempting to load security configuration
May 14, 2010 2:09:47 PM null null
AUDIT: chain.started
May 14, 2010 2:09:48 PM null null
WARNING: SIB_MESSAGE
May 14, 2010 2:09:48 PM null null
INFO: SOME_FUTURE_MESSAGES_SUPPRESSED_CWSIU0005
May 14, 2010 2:09:48 PM null null
AUDIT: chain.started
CWL RB5020I: Fri May 14 14:09:49 CDT 2010 : Job [84] has been submitted for execution.
CWL RB5020I: Fri May 14 14:09:49 CDT 2010 : Job [MAILJOB:00084] has been submitted for execution.
...
```

This is followed by the listing of the xJCL used both before and after substitution properties are applied and the log produced by the batch application. Finally, at the end of the output is the completion status of the job and the result code.

```
CWL RB5594I: [05/14/10 14:09:52:313 CDT] Step SendPromotionsStep execution is complete: ended normally
CWL RB1890I: [05/14/10 14:09:52:313 CDT] Unsubscribing from job cancel or stop subject:
BizGridJobCancel MAILJOB:00084
CWL RB3800I: [05/14/10 14:09:52:313 CDT] Job [MAILJOB:00084] ended normally.
CWL RB5596I: [05/14/10 14:09:52:313 CDT] Grid Execution Environment sequential step processing complete:
ended
CWL RB2250I: [05/14/10 14:09:52:313 CDT] Job setup manager bean is breaking down job: MAILJOB:00084
CWL RB5598I: [05/14/10 14:09:52:329 CDT] Removing job abstract resources
CWL RB5600I: [05/14/10 14:09:52:329 CDT] Removing job step status table entries
CWL RB2270I: [05/14/10 14:09:52:329 CDT] Job setup manager bean completed job MAILJOB:00084 breakdown
CWL RB5764I: [05/14/10 14:09:52:329 CDT] Job MAILJOB:00084 ended
CWL RB3880I: Job [MAILJOB:00084] ending status: RC=0
```

- ___3. **Before entering any other commands in the command prompt window.** Type the following command: **ECHO %ERRORLEVEL%**. This allows you to see that the return code from the job was returned by WSGrid to the caller. On Linux and Unix platforms the value will be stored in the \$@ shell variable.

```
C:\ClassMaterials\CG\WSGrid\working>ECHO %errorlevel%
0
```

3.2.2 Restarting a job using WSGrid

In this section we explore how WSGrid allows for submission of restartable jobs. First we must inject an error into the scenario. You will be working in two command prompt windows , both DB2CMD and the one you have been using for WSGrid in the previous section.

- ___1. Open a DB2 command window by typing **db2cmd** in any of the command prompt windows or in Window's **Start->Run..** dialog. You will use this window in both this and the next section. It will also be useful in other later labs so don't close it when you are done.
- ___2. In the db2 command prompt window, change the current directory using the following command:
cd C:\ClassMaterials\CG\checkpointRestart
- ___3. Enter the following command to run the script that modifies a customer entry to introduce the failure scenario:

db2 -tf injectInvalidCustomerData.sql

```

C:\ClassMaterials\CG\checkpointRestart>db2 -tf injectInvalidCustomerData.sql
SQL1024N  A database connection does not exist.  SQLSTATE=08003

Database Connection Information
Database server      = DB2/NT 9.7.0
SQL authorization ID = DB2ADMIN
Local database alias = MAILER

DB20000I  The SQL command completed successfully.
DB20000I  The SQL command completed successfully.

NAME                ADDRESS
SS                  CITY
                    STATE ZIPCODE
-----
EMAIL                ANNUALINCOME  CUSTOMERID  LASTOFFERDATE
PHONE

-----
Rhona F. Herring
                    -
                    Rochester
                    TX  41789

477-774-2159        18664    1236589  11/04/2009

1 record(s) selected.

DB20000I  The SQL command completed successfully.
DB21007E  End of file reached while reading the command.
C:\ClassMaterials\CG\checkpointRestart>

```


4. In the other command prompt window, launch WSGrid using the control and properties file for the normal job using the following command: **Notice the addition of restart.properties.**

c:\IBM\WebSphere\AppServer\bin\WSGrid.bat control.properties job.properties restart.properties

The restart.properties file gives the name of a non-existent file that will be created by WSGrid if a failure occurs. It will contain a property in the form restart-job=FAILED_JOB_ID. This is the way that WSGrid and its calling enterprise scheduler can identify the instance of the job that needs to be restarted.

After the job fails something like the following will be shown. The -12 return code indicates the job failed in a restartable state.

```
Caused by: java.lang.RuntimeException: Invalid Postal address for customer without email address,
CustomerID = 1236589
    at com.ibm.websphere.samples.IdentifyRecipientsStep.processRecord(IdentifyRecipientsStep.java:83)
    at
com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep.processRecord(GenericXDBa
tchStep.java:251)
    at
com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep.processJobStep(GenericXDB
atchStep.java:216)
    ... 19 more
CWLRB2280E: [05/14/10 14:12:28:782 CDT] [Long Running Job Container step execution failed] [Job
MAILJOB:00086] [Step IdentifyRecipientsStep]: com.ibm.websphere.batch.BatchContainerApplicationException:
CWLRB2280E: [Long Running Job Container step execution failed] [Job MAILJOB:00086] [Step
IdentifyRecipientsStep]: javax.ejb.TransactionRolledbackLocalException: ; nested exception is:
java.lang.RuntimeException: Unexpected error in batch loop
CWLRB5606I: [05/14/10 14:12:28:782 CDT] Destroying job step: IdentifyRecipientsStep
CWLRB5624I: [05/14/10 14:12:28:782 CDT] Rolling back step IdentifyRecipientsStep recordbased checkpoint.
User transaction status: STATUS MARKED ROLLBACK
CWLRB5602I: [05/14/10 14:12:28:782 CDT] Closing IdentifyRecipientsStep batch data stream: inputStream
CWLRB5602I: [05/14/10 14:12:28:782 CDT] Closing IdentifyRecipientsStep batch data stream: outputStream
CWLRB5604I: [05/14/10 14:12:28:782 CDT] Freeing IdentifyRecipientsStep batch data stream: inputStream
CWLRB5604I: [05/14/10 14:12:28:782 CDT] Freeing IdentifyRecipientsStep batch data stream: outputStream
CWLRB5606I: [05/14/10 14:12:28:798 CDT] Destroying job step: IdentifyRecipientsStep
CWLRB5604I: [05/14/10 14:12:28:798 CDT] Freeing IdentifyRecipientsStep batch data stream: inputStream
CWLRB5604I: [05/14/10 14:12:28:798 CDT] Freeing IdentifyRecipientsStep batch data stream: outputStream
CWLRB5594I: [05/14/10 14:12:28:813 CDT] Step IdentifyRecipientsStep execution is complete: ended
abnormally EJB transaction rolled back
CWLRB1890I: [05/14/10 14:12:28:813 CDT] Unsubscribing from job cancel or stop subject:
BizGridJobCancel MAILJOB:00086
CWLRB5596I: [05/14/10 14:12:28:813 CDT] Grid Execution Environment sequential step processing complete:
restartable
CWLRB5592I: [05/14/10 14:12:28:813 CDT] Execution complete: restartable
CWLRB3880I: Job [MAILJOB:00086] ending status: RC=-12
```

- ___5. **Before entering any other commands in the command prompt window.** Type the following command: **ECHO %ERRORLEVEL%**. This allows you to see that the return code from the job was returned by WSGrid to the caller, in the case a -12. On Linux and Unix platforms the value will be stored in the \$@ shell variable.

```
C:\ClassMaterials\CG\WSGrid\working>ECHO %errorlevel%
-12
```

- ___6. In the DB2 command prompt window in the same directory as in the previous section, enter the following command. It will modify a customer entry to correct the failure encountered in the previous section.

db2 -tf removeInvalidCustomerData.sql

```

C:\ClassMaterials\CG\checkpointRestart>db2 -tf removeInvalidCustomerData.sql
SQL1024N  A database connection does not exist.  SQLSTATE=08003

Database Connection Information
Database server      = DB2/NT 9.7.0
SQL authorization ID = DB2ADMIN
Local database alias = MAILER

DB20000I  The SQL command completed successfully.
DB20000I  The SQL command completed successfully.

NAME                                     ADDRESS
SS                                     CITY
                                     STATE ZIPCODE
-----
EMAIL ANNUALINCOME CUSTOMERID LASTOFFERDATE
PHONE
-----
Rhona E. Herring                                     159-4
689 Aliquet Ave                                     Rochester
herring@xyz.edu                                     TX 41789
477-774-2139
18664 1236589 11/04/2009
1 record(s) selected.

DB20000I  The SQL command completed successfully.
DB21007E  End of file reached while reading the command.
C:\ClassMaterials\CG\checkpointRestart>_

```

- ___7. Back in the WSGrid command prompt window launch WSGrid using the control and the restart.properties file. The job properties file is not needed this time since the original submission properties are contained in the restartable instance of the job.

c:\IBM\WebSphere\AppServer\bin\WSGrid.bat control.properties restart.properties

The job should complete successfully this time with a result code 0.

```
CWLRB2600I: [05/14/10 14:13:16:501 CDT] Job [MAILJOB:00086] Step [SendPromotionsStep] completed normally
rc=0.
CWLRB5594I: [05/14/10 14:13:16:532 CDT] Step SendPromotionsStep execution is complete: ended normally
CWLRB1890I: [05/14/10 14:13:16:532 CDT] Unsubscribing from job cancel or stop subject:
BizGridJobCancel_MAILJOB:00086
CWLRB3800I: [05/14/10 14:13:16:532 CDT] Job [MAILJOB:00086] ended normally.
CWLRB5596I: [05/14/10 14:13:16:532 CDT] Grid Execution Environment sequential step processing complete:
ended
CWLRB2250I: [05/14/10 14:13:16:532 CDT] Job setup manager bean is breaking down job: MAILJOB:00086
CWLRB5598I: [05/14/10 14:13:16:532 CDT] Removing job abstract resources
CWLRB5600I: [05/14/10 14:13:16:548 CDT] Removing job step status table entries
CWLRB2270I: [05/14/10 14:13:16:563 CDT] Job setup manager bean completed job MAILJOB:00086 breakdown
CWLRB5764I: [05/14/10 14:13:16:563 CDT] Job MAILJOB:00086 ended
CWLRB3880I: Job [MAILJOB:00086] ending status: RC=0
```

- ___8. **Before entering any other commands in the command prompt window.** Type the following command: **ECHO %ERRORLEVEL%**. This allows you to see that the return code from the job was returned by WSGrid to the caller. On Linux and Unix platforms the value will be stored in the **\$@** shell variable.

```
C:\ClassMaterials\CG\WSGrid\working>ECHO %errorlevel%
0
```

3.2.3 Submit a parallel job from the job repository.

- __1. Return to the command prompt window and the directory
C:\ClassMaterials\CG\WSGrid\working
- __2. Launch WSGrid using the control and properties file for the parallel job using the following command:

c:\IBM\WebSphere\AppServer\bin\WSGrid.bat control.properties pjmljob.properties

The log output for the top level job and all of the subjobs will be returned to standard output as shown below. The subjobs and the top level job should finish successfully.

```
...
System.out: [05/14/10 14:14:25:173 CDT] ***** End ParallelJob:00087:00089 log *****
System.out: [05/14/10 14:14:25:173 CDT] ***** Begin ParallelJob:00087:00088 log *****
... subjob log
System.out: [05/14/10 14:14:25:173 CDT] ***** End ParallelJob:00087:00088 log *****
CWLRB5610I: [05/14/10 14:14:25:313 CDT] Firing Step1 results algorithm
com.ibm.wsspi.batch.resultsalgorithms.jobsum: [RC 0] [jobRC 0]
CWLRB5624I: [05/14/10 14:14:25:313 CDT] Stopping step Step1 timebased checkpoint. User transaction status:
STATUS_ACTIVE
CWLRB2600I: [05/14/10 14:14:25:391 CDT] Job [ParallelJob:00087] Step [Step1] completed normally rc=0.
CWLRB5594I: [05/14/10 14:14:25:423 CDT] Step Step1 execution is complete: ended normally
CWLRB1890I: [05/14/10 14:14:25:438 CDT] Unsubscribing from job cancel or stop subject:
BizGridJobCancel_ParallelJob:00087
CWLRB3800I: [05/14/10 14:14:25:438 CDT] Job [ParallelJob:00087] ended normally.
CWLRB5596I: [05/14/10 14:14:25:438 CDT] Grid Execution Environment sequential step processing complete:
ended
CWLRB2250I: [05/14/10 14:14:25:438 CDT] Job setup manager bean is breaking down job: ParallelJob:00087
CWLRB5598I: [05/14/10 14:14:25:454 CDT] Removing job abstract resources
CWLRB5600I: [05/14/10 14:14:25:470 CDT] Removing job step status table entries
CWLRB2270I: [05/14/10 14:14:25:485 CDT] Job setup manager bean completed job ParallelJob:00087 breakdown
CWLRB5764I: [05/14/10 14:14:25:485 CDT] Job ParallelJob:00087 ended
CWLRB3880I: Job [ParallelJob:00087] ending status: RC=0
```

- __3. **Before entering any other commands in the command prompt window.** Type the following command: **ECHO %ERRORLEVEL%.** This allows you to see that the return code from the job was returned by WSGrid to the caller. On Linux and Unix platforms the value will be stored in the **\$@** shell variable.

```
C:\ClassMaterials\CG\WSGrid\working>ECHO %errorlevel%
0
```

3.2.4 Restarting a parallel job using WSGrid

In this section we explore how WSGrid allows for submission of restartable parallel jobs. First we must inject an error into the scenario. You will be working in two command prompt windows , both DB2CMD and the one you have been using for WSGrid in the previous section.

- ___1. Open a DB2 command window by typing **db2cmd** in any of the command prompt windows or in Window's **Start->Run..** dialog. You will use this window in both this and the next section. It will also be useful in other later labs so don't close it when you are done.
- ___2. In the db2 command prompt window, change the current directory using the following command:
cd C:\ClassMaterials\CG\checkpointRestart
- ___3. Enter the following command to run the script that modifies a customer entry to introduce the failure scenario:

db2 -tf injectInvalidCustomerData.sql

```

C:\ClassMaterials\CG\checkpointRestart>db2 -tf injectInvalidCustomerData.sql
SQL1024N  A database connection does not exist.  SQLSTATE=08003

Database Connection Information
Database server      = DB2/NT 9.7.0
SQL authorization ID = DB2ADMIN
Local database alias = MAILER

DB20000I  The SQL command completed successfully.
DB20000I  The SQL command completed successfully.

NAME                                ADDRESS                                CITY                                STATE  ZIPCODE
-----                                -
Rhona F. Herring                    Rochester                                TX    41789

EMAIL                                PHONE                                ANNUALINCOME  CUSTOMERID  LASTOFFERDATE
-----                                -
477-774-2159                        18664      1236589  11/04/2009

1 record(s) selected.

DB20000I  The SQL command completed successfully.
DB21007E  End of file reached while reading the command.
C:\ClassMaterials\CG\checkpointRestart>

```

- __4. In the other command prompt window, launch WSGrid using the control and properties file for the parallel job using the following command: **Notice the addition of restart.properties.**

```
c:\IBM\WebSphere\AppServer\bin\WSGrid.bat control.properties pjmljob.properties restart.properties
```

The restart.properties file gives the name of a non-existent file that will be created by WSGrid in if a failure occurs. It will contain a property in the form restart-job=FAILED_JOB_ID. This is the way that WSGrid and its calling enterprise scheduler can identify the instance of the top level job that needs to be restarted.

After the job fails something like the following will be shown. The -12 return code indicates the top level job failed in a restartable state.

```
CWLRB5606I: [05/14/10 14:18:52:766 CDT] Destroying job step: Step1
System.out: [05/14/10 14:18:52:845 CDT] Calling
com.ibm.wsspi.batch.router.SynchronizationRouter.afterCompletion("ParallelJob:00092",ROLLBACK)
System.out: [05/14/10 14:18:52:845 CDT]
MailerTXSynchronization.afterCompletion(ParallelJob:00092,ROLLBACK) called.
CWLRB5608I: [05/14/10 14:18:52:845 CDT] Job step Step1 destroy completed with rc: -12
CWLRB5624I: [05/14/10 14:18:52:845 CDT] Rolling back step Step1 timebased checkpoint. User transaction
status: STATUS_ACTIVE
CWLRB5607W: [05/14/10 14:18:52:845 CDT] Job step Step1 destroy completed with rc: -12 which is within the
system application return code range
CWLRB3860W: [05/14/10 14:18:52:860 CDT] Job [ParallelJob:00092] ended abnormally [and is restartable].
CWLRB5594I: [05/14/10 14:18:52:907 CDT] Step Step1 execution is complete: ended abnormally
CWLRB1890I: [05/14/10 14:18:52:907 CDT] Unsubscribing from job cancel or stop subject:
BizGridJobCancel_ParallelJob:00092
CWLRB5596I: [05/14/10 14:18:52:907 CDT] Grid Execution Environment sequential step processing complete:
restartable
CWLRB5592I: [05/14/10 14:18:52:907 CDT] Execution complete: restartable
CWLRB3880I: Job [ParallelJob:00092] ending status: RC=-12
```

- __5. **Before entering any other commands in the command prompt window.** Type the following command: **ECHO %ERRORLEVEL%.** This allows you to see that the return code from the top level job was returned by WSGrid to the caller, in the case a -12. On Linux and Unix platforms the value will be stored in the \$@ shell variable.

```
C:\ClassMaterials\CG\WSGrid\working>ECHO %errorlevel%
-12
```

- ___6. In the DB2 command prompt window in the same directory as in the previous section, enter the following command. It will modify a customer entry to correct the failure encountered in the previous section.

db2 -tf removeInvalidCustomerData.sql

```

C:\ClassMaterials\CG\checkpointRestart>db2 -tf removeInvalidCustomerData.sql
SQL1024N  A database connection does not exist.  SQLSTATE=08003

Database Connection Information
Database server      = DB2/NT 9.7.0
SQL authorization ID = DB2ADMIN
Local database alias = MAILER

DB200000I  The SQL command completed successfully.
DB200000I  The SQL command completed successfully.

NAME                                               ADDRE
SS                                               CITY
                                               STATE ZIPCODE
-----
EMAIL      ANNUALINCOME  CUSTOMERID  LASTOFFERDATE
PHONE
-----
Rhona E. Herring
689 Aliquet Ave                                     159-4
                                               Rochester
TX  41789
herring@xyz.edu
477-74-2159
18664      1236589  11/04/2009

1 record(s) selected.

DB200000I  The SQL command completed successfully.

DB21007E  End of file reached while reading the command.
C:\ClassMaterials\CG\checkpointRestart>

```

- ___7. Back in the WSGrid command prompt window launch WSGrid using the control and the restart.properties file. The pjml job properties file is not needed this time since the original submission properties are contained in the restartable instance of the job.

c:\IBM\WebSphere\AppServer\bin\WSGrid.bat control.properties restart.properties

The job should complete successfully this time with a result code 0.

```

CWLRB2600I: [05/14/10 14:19:32:345 CDT] Job [ParallelJob:00092] Step [Step1] completed normally rc=0.
CWLRB5594I: [05/14/10 14:19:32:516 CDT] Step Step1 execution is complete: ended normally
CWLRB1890I: [05/14/10 14:19:32:532 CDT] Unsubscribing from job cancel or stop subject:
BizGridJobCancel_ParallelJob:00092
CWLRB3800I: [05/14/10 14:19:32:532 CDT] Job [ParallelJob:00092] ended normally.
CWLRB5596I: [05/14/10 14:19:32:563 CDT] Grid Execution Environment sequential step processing complete:
ended
CWLRB2250I: [05/14/10 14:19:32:563 CDT] Job setup manager bean is breaking down job: ParallelJob:00092
CWLRB5598I: [05/14/10 14:19:32:579 CDT] Removing job abstract resources
CWLRB5600I: [05/14/10 14:19:32:579 CDT] Removing job step status table entries
CWLRB2270I: [05/14/10 14:19:32:595 CDT] Job setup manager bean completed job ParallelJob:00092 breakdown
CWLRB5764I: [05/14/10 14:19:32:595 CDT] Job ParallelJob:00092 ended
CWLRB3880I: Job [ParallelJob:00092] ending status: RC=0

```

- ___8. **Before entering any other commands in the command prompt window.** Type the following command: **ECHO %ERRORLEVEL%**. This allows you to see that the return code from the job was returned by WSGrid to the caller. On Linux and Unix platforms the value will be stored in the \$@ shell variable.

```
C:\ClassMaterials\CG\WSGrid\working>ECHO %errorlevel%  
0
```

Keep both of the command prompt windows open from this section they will be useful in the next section.

3.2.5 Automating submission and restart – a simple example

In this section you will use a .bat script written to demonstrate how a computer process such as an enterprise scheduler or even a simple script can drive WSGrid submissions and perform automatic restarts. Concepts such as mapping an arbitrary job ID to the compute grid job and retaining job logs are also demonstrated.

___1. In either notepad or Notepad++ open the script file:

C:\ClassMaterials\CG\WSGrid\working\wsgrid_demo.bat

Observe how the normal and restart WSGrid invocation styles are used to either submit a job or restart a job and how the -12 result code is used to determine that a restart is called for. The following is an excerpt from the script that performs some of these checks.

```
REM =====
REM This is the initial submission of the job.
ECHO Submitting job %UNIQUE_JOB_ID% using the following job properties:
TYPE %UNIQUE_JOB_ID%.job.props
1>>%UNIQUE_JOB_ID%.job.log 2>&1 CALL %WSGRIDCMD% control.properties %UNIQUE_JOB_ID%.job.props
%UNIQUE_JOB_ID%.restart.props
SET RC=%ERRORLEVEL%
IF %RC% EQU -12 GOTO RESTART
IF %RC% GEQ 0 GOTO SUCCESS
GOTO FAIL

REM =====
REM Submission of a restartable job
:RESTART
ECHO Job %UNIQUE_JOB_ID% failed with RC= %RC% and is restartable. Press enter to restart or ctrl-C to exit
PAUSE
ECHO Restarting job %UNIQUE JOB ID% using the following WebSphere Compute Grid Job Scheduler ID:
TYPE %UNIQUE_JOB_ID%.restart.props
1>>%UNIQUE_JOB_ID%.job.log 2>&1 CALL %WSGRIDCMD% control.properties %UNIQUE_JOB_ID%.restart.props
SET RC=%ERRORLEVEL%
IF %RC% EQU -12 GOTO RESTART
IF %RC% GEQ 0 GOTO SUCCESS
GOTO FAIL
```

___2. In the WSGrid command prompt window used in the previous sections verify that you are still in the directory **C:\ClassMaterials\CG\WSGrid\working**.

Enter the command **wsgrid_demo.bat** without any parameters. This will provide the following usage:

```
Usage:
WSGRID_DEMO job_repository_name unique_job_id [subst_prop_name=subst_prop_value]
Where:
job_repository_name      = Name of job definition in job scheduler repository.
unique_job_id           = Unique ID for job assigned by caller.
[subst_prop_name=subst_prop_value] = Optional occurrences of name value pairs assigning values
                                for substitution properties defined in the job definition.
Purpose:
This is a simple script that provides a rudimentary demonstration of using WSGrid to submit
a job to the WebSphere Compute Grid Job Scheduler. It does many of the things an enterprise scheduler
might do while making such submissions.
These include:
1) Accepting a unique ID provided by the enterprise scheduler to correlate this job with its
   logical representation in the enterprise scheduler.
2) Build the required job invocation properties and stage them for submission.
3) Archive or clean up the artifacts from the job execution.
4) Restart a job that has failed in a restartable state using WebSphere's assigned job id.
```

- __3. Next submit a normal job by entering the following command.
wsgrid_demo.bat NormalWSGridJob AAA111
 Where AAA111 is an arbitrary unique identifier you have made up for the this invocation.
 The result should be as follows:

```
Submitting job AAA111 using the following job properties:
repository-job=NormalWSGridJob
substitution-prop.DEBUG=false
Job AAA111 Succeeded with return code 0
```

- __4. Using the DM2CMD window run the **injectInvalidCustomerData.sql** script to introduce failure as you did in the previous sections.
 Run the command again with a different unique ID as follows:
wsgrid_demo.bat NormalWSGridJob BBB222

```
Submitting job BBB222 using the following job properties:
repository-job=NormalWSGridJob
substitution-prop.DEBUG=false
Job BBB222 failed with RC= -12 and is restartable. Press enter to restart or ctrl-C to exit
Press any key to continue . . .
```

- __5. Using the DM2CMD window run the **removeInvalidCustomerData.sql** script to cleanup failure scenario and then pres enter in response to the **wsgrid_demo.bat** prompt above.
 The job should be restarted and the result should appear as follows:

```
Submitting job BBB222 using the following job properties:
repository-job=NormalWSGridJob
substitution-prop.DEBUG=false
Job BBB222 failed with RC= -12 and is restartable. Press enter to restart or ctrl-C to exit
Press any key to continue . . .
Restarting job BBB222 using the following WebSphere Compute Grid Job Scheduler ID:
restart-job=MAILJOB:00100Job BBB222 Succeeded with return code 0
```

- __6. Look in the **completed_job** subdirectory to find the job logs and submission properties retained from the previous runs.
- __7. You can repeat these using **ParallelWSGridJob** as the **job_repository_name** to see that the same approach works for parallel jobs.

Appendix A. Notices

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

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

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

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

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have

been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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

COPYRIGHT LICENSE:

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

Appendix B. Trademarks and copyrights

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

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
<i>System z</i>	<i>Tivoli</i>	<i>WebSphere</i>	<i>Workplace</i>	<i>System p</i>	

Adobe, Acrobat, Portable Document Format (PDF), and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. See Java Guidelines

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

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

ITIL is a registered trademark and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

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



© Copyright IBM Corporation 2011.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

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. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at ibm.com/legal/copytrade.shtml

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



Please Recycle
