IBM InfoSphere DataStage and QualityStage
Version 9 Release 1

Guide to Integrating Java Code
IBM InfoSphere DataStage and QualityStage
Version 9 Release 1

Guide to Integrating Java Code

IBM
Note

Before using this information and the product that it supports, read the information in “Notices and trademarks” on page 61.
Chapter 1. Integrating Java Code (Java Integration stage)

When you use IBM® InfoSphere® DataStage® to invoke Java code, you can choose from a collection of connectivity options. For most new jobs, use the Java Integration stage, which offers better functionality and performance.

The Java Integration stage can be used in the following topologies:
- As a source with one or more output links
- As a target with one or more input links and zero or more reject links
- As a transformer with one or more input links and one or more output or reject links
- As a lookup stage with one reference output link

The Java Integration stage is one of several different stages that invokes Java code. In addition to the Java Integration stage, the following stages are available:
- Java Client stage
- Java Transformer stage

If you want to integrate Java code into a server job, you must use one of the older stages instead of the Java Integration stage.

If you have jobs that use the older stages and want to use the Java Integration stage, use the Connector Migration Tool to migrate jobs to use connectors.
Chapter 2. Writing Java code to use in jobs (Java Integration stage)

You can use the Java Integration stage to integrate your code into your job design by writing your Java code using the Java Integration stage API. The Java Integration stage API defines interfaces and classes for writing Java code which can be invoked from within InfoSphere DataStage and QualityStage parallel jobs.

Related information:

[Java Pack API]
Click the link to see the Javadoc information for Java Pack API. Also see the Javadoc information for the Java Integration stage API.

Setting up your development environment (Java Integration stage)

You need to set up your development environment before creating your Java code.

Procedure

1. Install Java 6 SDK.
2. Copy ccjava-api.jar to your workspace as you need the file to compile your Java code. The ccjava-api.jar file is available in the following locations:
   a. <ISDIR>/Server/DSComponents/bin, if you are using an Information Server engine tier machine.
   b. <ISDIR>/Clients/Samples/Connectors/JavaIntegration_Samples.zip, if you are using an Information Server client tier machine.

Installing API Documents and Samples (Java Integration stage)

The Java Integration stage API documents and samples are installed on the system where you installed the Information Server Client tier.

The following two API documents are installed in the location <ISInstall directory>Clients\Samples\Connectors:

- JavaIntegration_API_Document.zip - This file contains the Javadoc for Java Integration stage API.
- JavaIntegration_Samples.zip - This file contains the API samples for the Java Integration stage API.

Implementing abstract methods of the Processor class (Java Integration stage)

Your Java code must implement a subclass of the Processor class. The Processor class consists of methods that are invoked by the Java Integration stage. When a job that includes the Java Integration stage starts, the stage instantiates your Processor class and calls the logic within your Processor implementations.

The Processor class provides the following list of methods that the Java Integration stage can call to interact with your Java code at job execution time or at design-time:

- getCapabilities()
At minimum, your Java code must implement the following two abstract methods.

- public abstract boolean validateConfiguration(Configuration configuration, boolean isRuntime) throws Exception;
- public abstract void process() throws Exception;

The following example shows the simple peek stage implementation that prints record column values to the job log which can be viewed in Director client. It assumes single input link.

```java
public class SimplePeek extends Processor {
    private InputLink m_inputLink;

    public boolean validateConfiguration(
            Configuration configuration, boolean isRuntime) throws Exception
    {
        if (configuration.getInputLinkCount() != 1)
        {
            // this sample code assumes stage has 1 input link.
            return false;
        }

        m_inputLink = configuration.getInputLink(0);

        return true;
    }

    public void process() throws Exception
    {
        do
        {
            InputRecord inputRecord = m_inputLink.readRecord();
            if (inputRecord == null)
            {
                // No more input. Your code must return from process() method.
                break;
            }

            for (int i = 0; i < m_inputLink.getColumnCount(); i++)
            {
                Object value = inputRecord.getValue(i);
                Logger.information(value.toString());
            }
        }
    }
}
```
The Java Integration stage calls the `validateConfiguration()` method to specify the current configuration (number and types of links), and the values for the user properties. Your Java code must validate a given configuration and user properties and return `false` to Java Integration stage if there are problems with them. In the previous example, since this code assumes a stage that has single input link, it checks the number of input links and returns `false` if the stage configuration does not meet this requirement.

```java
if (configuration.getInputLinkCount() != 1)
{
    // this sample code assumes stage has 1 input link.
    return false;
}
```

The `Configuration` interface defines methods that are used to get the current stage configuration (number and types of links), and the values for the user properties. The `getInputLinkCount()` method is used to get the number of input links connected to this stage.

If stage configuration is accepted by your Java code, it saves the reference to an `InputLink` object for subsequent processing, and returns `true` to the Java Integration stage.

```java
m_inputLink = configuration.getInputLink(0);
return true;
```

After the stage configuration is verified by your Java code, you can interact with the stages connected in your job. The `process()` method is an entry point for processing records from the input link or to the output link. When a row is available on any of the stage input links (if any and whatever the number of output links is), the Java Integration stage calls this method, if the job does not end. Your Java code must consume all rows from the stage input links.

By calling the `readRecord()` method of the `InputLink` interface, your Java code can consume a row from the input link. It returns an object that implements the `InputRecord` interface. The `InputRecord` interface defines methods that are used to get column data from a consumed row record.

```java
InputRecord inputRecord = m_inputLink.readRecord();
if (inputRecord == null)
{
    // No more input. Your code must return from process() method.
    break;
}
```

After your Java code consumes a row record from the stage input link, your Java code can get the column record values by calling the `getValue(int columnIndex)` method of the `InputRecord` interface. The `getColumnCount()` in `InputLink` returns the number of columns that exist in this input link.

```java
for (int i = 0; i < m_inputLink.getColumnCount(); i++)
{
    Object value = inputRecord.getValue(i);
}
```

Finally, each column value is written to the job log by calling the `information()` method of the `Logger` class. The `Logger` class allows your Java code to write the
data to job log with specified log levels. The following code writes the string representation of each column value to a job log.

Logger.information(value.toString());

---

**Compiling the Java code (Java Integration stage)**

After creating your Java code, you must compile the Java code and optionally, you may create a JAR file for deployment.

**Procedure**

1. Compile your Java code with ccjava-api.jar, using the following command:
   
   `javac -cp ..\jars\ccjava-api.jar samples\SimplePeek.java`

2. Create a jar file for deployment, using the following command:

   `jar -cvf ..\jars\samples.jar samples\SimplePeek.class`

---

**Running the Java code on the Parallel Engine (Java Integration stage)**

Your Java code is invoked by one or more Java VM instances in the InfoSphere DataStage parallel engine environment. The conductor is an initial DataStage process that creates a single Java VM instance and loads your Java code and other Java-based Connector stage code in your job. The processors on the player nodes are forked per each stage in the job and are the actual processes that are associated with stages. A Java VM instance is created for each player process that is associated with your Java Integration stage where your Java code runs.

---

**Accessing Stage Configuration (Java Integration stage)**

An instance of the `Configuration` interface defines the current stage configuration, such as number and type of links and the values for the user-defined properties that are specified in the job.

To see the details of the available methods that the `Configuration` interface provides, see the Javadoc information for the Java Integration stage API.

**Methods for accessing link configurations**

- `getLinks()`
- `getInputLinks()`
- `getInputLinkCount()`
- `getInputLinks()`
- `getOutputLink()`
- `getOutputLinkCount()`
- `getOutputLinks()`
- `getRejectOutputLink()`
- `getRejectLinkCount()`
- `getStreamOutputLink()`
- `getStreamOutputLinkCount()`

**Methods for accessing node configurations**

- `getNodeCount()`
- `getNodeNumber()`
Method for accessing user-defined stage properties

- `getUserProperties()`

---

### Declaring the Capabilities of the Java code (Java Integration stage)

The `Capabilities` class defines the capabilities of your Java code by encapsulating a list of attributes and parameters.

The following is a list of the available methods that the `Capabilities` class provides. For details on the methods see the Javadoc information for the Java Integration stage API documentation.

- `getMinimumInputLinkCount()`  
- `getMaximumInputLinkCount()`  
- `getMinimumOutputStreamLinkCount()`  
- `getMaximumOutputStreamLinkCount()`  
- `getMinimumRejectLinkCount()`  
- `getMaximumRejectLinkCount()`  
- `isWaveGenerator()`  
- `getColumnTransferBehavior()`  
- `setMinimumInputLinkCount()`  
- `setMaximumInputLinkCount()`  
- `setMinimumOutputStreamLinkCount()`  
- `setMaximumOutputStreamLinkCount()`  
- `setMinimumRejectLinkCount()`  
- `setMaximumRejectLinkCount()`  
- `setIsWaveGenerator()`  
- `setColumnTransferBehavior()`

Right after instantiating your `Processor` code, the Java Integration stage invokes the `getCapabilities()` method in your `Processor` code to get its associated `Capabilities` object to determine whether your Java code can be run in the current job design.

By overriding the `getCapabilities()` method in the `Processor` class, your Java code can customize the values of the capabilities to address your Java code, and pass it to the Java Integration stage.

The following example shows that your Java code only accepts the case of single input link.

```java
public Capabilities getCapabilities()
{
    Capabilities capabilities = new Capabilities();
    capabilities.setMinimumInputLinkCount(1);
    capabilities.setMaximumInputLinkCount(1);
    capabilities.setMaximumOutputStreamLinkCount(0);
    capabilities.setMaximumRejectLinkCount(0);
    return capabilities;
}
```

The job ends if the current job design does not fit the specified capabilities.

The following code provides the functionality which is equivalent to the first example. The Java Integration stage will compare the number of links attached to
the stage with the limits specified by the implementation of the getCapabilities() method. If the number of links it outside of the specified bounds, then the Java Integration stage will send an appropriate message to the job log and will abort the job.

```java
package samples;
import com.ibm.is.cc.javastage.api.*;

public class ReworkedSimplePeek extends Processor {
    private InputLink m_inputLink;

    public Capabilities getCapabilities() {
        Capabilities capabilities = new Capabilities();
        capabilities.setMinimumInputLinkCount(1);
        capabilities.setMaximumInputLinkCount(1);
        capabilities.setMaximumOutputStreamLinkCount(0);
        capabilities.setMaximumRejectLinkCount(0);
        return capabilities;
    }

    public boolean validateConfiguration(Configuration configuration, boolean isRuntime) throws Exception {
        m_inputLink = configuration.getInputLink(0);
        return true;
    }

    public void process() throws Exception {
        do {
            InputRecord inputRecord = m_inputLink.readRecord();
            if (inputRecord == null) {
                // No more input. Your code must return from process() method.
                break;
            }

            for (int i = 0; i < m_inputLink.getColumnCount(); i++) {
                Object value = inputRecord.getValue(i);
                Logger.information(value.toString());
            }
        } while (true);
    }
}
```

**Reading Records from Input Link (Java Integration stage)**

The `InputLink` interface is an extension of the `Link` interface. It defines methods that are used to interact with corresponding stage input link. The instances of an `InputLink` are available in the `Configuration` object that is provided as an argument of the `validateConfiguration()` method.

**Methods provided by Link interface**
- `getColumn()`
- `getColumnCount()`
- `getColumnMetadata()`
- `getLinkIndex()`
• getUserProperties()
• subtractColumnList()

Methods provided by InputLink interface
• GetAssociatedRejectLink()
• readRecord()

By calling the readRecord() method of an InputLink interface, your Java code can consume a row from input link. It returns an object that implements the InputRecord interface.

```java
InputRecord inputRecord = m_inputLink.readRecord();
```

The InputRecord interface is an extension of the Record interface. It defines methods that are used to get column data from a consumed row record.

Methods provided by InputRecord interface
• get0bject()
• getValue(String columnName)
• getValue(int columnIndex)

The following example shows how to retrieve the value corresponding to a given column index “i” in this record.

```java
Object value = inputRecord.getValue(i);
```

You can also retrieve the value by specifying the column name like below.

```java
Object value = inputRecord.getValue("name");
```

---

**Writing Records to Output Link (Java Integration stage)**

To write records to output link, your Java code needs to instantiate an OutputRecord object by using the getOutputRecord() method of the OutputLink interface.

The following example shows the simple transformer stage implementations that converts string texts in the consumed record to upper-case, and then write it to an output link.

```java
package samples;
import com.ibm.is.cc.javastage.api.*;

public class ToUpperTransformer extends Processor
{
    private InputLink m_inputLink;
    private OutputLink m_outputLink;

    public Capabilities getCapabilities()
    {
        Capabilities capabilities = new Capabilities();
        // Set minimum number of input links to 1
        capabilities.setMinimumInputLinkCount(1);
        // Set maximum number of input links to 1
        capabilities.setMaximumInputLinkCount(1);
        // Set minimum number of output stream links to 1
        capabilities.setMinimumOutputStreamLinkCount(1);
        // Set maximum number of output stream links to 1
        capabilities.setMaximumOutputStreamLinkCount(1);
        // Set maximum number of reject links to 1
    }

    public void process() throws Exception
    {
        // Process the input record
        InputRecord inputRecord = m_inputLink.readRecord();
        Object value = inputRecord.getValue("name");
        // Convert the value to upper-case
        String upperCaseValue = value.toString().toUpperCase();

        // Write the output record to the output link
        OutputRecord outputRecord = m_outputLink.getOutputRecord();
        outputRecord.writeObject(upperCaseValue);
    }
}
```
capabilities.setMaximumRejectLinkCount(0);  
return capabilities;
}

public boolean validateConfiguration(
    Configuration configuration, boolean isRuntime)
    throws Exception
{
    // Specify current link configurations.
    m_inputLink = configuration.getInputLink(0);
    m_outputLink = configuration.getOutputLink(0);

    return true;
}

public void process() throws Exception
{
    OutputRecord outputRecord = m_outputLink.getOutputRecord();

    do
    {
        InputRecord inputRecord = m_inputLink.readRecord();
        if (inputRecord == null)
        {
            // No more input
            break;
        }

        for (int i = 0; i < m_inputLink.getColumnCount(); i++)
        {
            Object value = inputRecord.getValue(columnIndex);
            if (value instanceof String)
            {
                String str = (String)value;
                value = str.toUpperCase();
            }
            outputRecord.setValue(i, value);
        }

        m_outputLink.writeRecord(outputRecord);
    }
    while (true);
}

To write records to an output link, your Java code needs to instantiate an
OutputRecord object by using the getOutputRecord() method of the OutputLink
interface.

OutputRecord outputRecord = m_outputLink.getOutputRecord();

Methods provided by OutputRecord interface
• putObject()
• setValue(String columnName, Object value)
• setValue(int columnIndex, Object value)
• setValueAsString(String columnName, String value)
• setValueAsString(int columnIndex, String value)
• copyColumnsFromInputRecord(InputRecord inputRecord)
• getOfLink()

After you instantiate the OutputRecord object, you can then set the value for each
column by using the setValue(String columnName, Object value) or the
setValue(int columnIndex, Object value) methods of the OutputRecord interface.
The following example shows how to set the value to the column corresponding to a given column index “i”.
```
outputRecord.setValue(i, value);
```

You can also set the value by specifying the column name as follows:
```
outputRecord.setValue("name", value);
```

Finally, your Java code writes this output record by calling the `writeRecord()` method of the OutputLink interface. The instance of the OutputLink is available in the Configuration object that is provided as argument of the validateConfiguration() method.
```
m_outputLink.writeRecord(outputRecord);
```

**Methods provided by OutputLink interface**
- `getOutputRecord()`
- `getOutputRecord(InputRecord)`
- `getRejectRecord(InputRecord)`
- `writeRecord()`
- `writeRecord(RejectRecord)`
- `writeWaveMarker()`
- `isRcpEnabled()`

**Rejecting Records (Java Integration stage)**

You might want to reject the records coming from an input link since the input record does not meet the requirements of your Java code. In this case, you might consider using a reject link in the job design and write the rejected data to this link.

When your Java code needs to write the record to a reject link, your Java code must call the `getAssociatedRejectLink()` method of the InputLink interface to get an instance of the OutputLink associated with the input link in the job design.
```
OutputLink m_rejectLink = m_inputLink.getAssociatedRejectLink();
```

Following is a list of methods provided by a RejectRecord interface:
- `setErrorText()`
- `setErrorCode()`
- `getOflink()`

Similar to the case of writing records to an output link, your Java code must instantiate a RejectRecord object by using the `getRejectRecord()` method of the OutputLink interface. Your Java code must specify the InputRecord object to be rejected.
```
RejectRecord rejectRecord = m_rejectLink.getRejectRecord(inputRecord);
```

A reject link in your job design might have the additional columns "ERRORTEXT" and "ERRORCODE". Your Java code can set the values for these additional columns by using the `setErrorText()` and the `setErrorCode()` methods of the RejectRecord interface.
```
rejectRecord.setErrorText("Name field contains *");
rejectRecord.setErrorCode(123);
```
Finally, you can write this reject record to the corresponding reject link by using the `writeRecord(RejectRecord)` method of the `OutputLink` interface.

```
m_rejectLink.writeRecord(rejectRecord);
```

**Data Types (Java Integration stage)**

InfoSphere DataStage supports a set of data types that are different from Java data types.

In an output link, where DataStage columns are set from the Java data types that are produced by the Java Integration stage, the Java Integration stage converts the Java data types to InfoSphere DataStage data types. Conversely, in an input link, where Java Bean properties or columns are set from the DataStage columns, the InfoSphere DataStage data types are converted to Java data types.

**Data type conversions from DataStage to Java data types (Java Integration stage)**

In an input link where Java types are set from DataStage columns consumed by the Java Integration stage, Java Integration stage converts InfoSphere DataStage data types to Java data types.

The following table shows the mapping rules between InfoSphere DataStage data types and Java data types.

**Table 1. InfoSphere DataStage data types and their equivalent Java data types**

<table>
<thead>
<tr>
<th>InfoSphere DataStage data types</th>
<th>Java data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigInt</td>
<td>java.math.BigInteger</td>
</tr>
<tr>
<td>Binary</td>
<td>byte[]</td>
</tr>
<tr>
<td>Bit</td>
<td>int/java.lang.Integer or boolean/</td>
</tr>
<tr>
<td></td>
<td>java.lang.Boolean</td>
</tr>
<tr>
<td><strong>Note:</strong> boolean/java.lang.Boolean is only applicable for Java bean or UDF case</td>
<td></td>
</tr>
<tr>
<td>Char</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>Date</td>
<td>java.sql.Date</td>
</tr>
<tr>
<td>Decimal</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>Double</td>
<td>double/java.lang.Double</td>
</tr>
<tr>
<td>Float</td>
<td>float/java.lang.Float</td>
</tr>
<tr>
<td>Integer</td>
<td>long/java.lang.Long</td>
</tr>
<tr>
<td>LongNVarChar</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>LongVarBinary</td>
<td>byte[]</td>
</tr>
<tr>
<td>LongVarChar</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>NChar</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>Numeric</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>NVarChar</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>Real</td>
<td>java.lang.Float</td>
</tr>
<tr>
<td>SmallInt</td>
<td>java.lang.Integer</td>
</tr>
<tr>
<td>Time</td>
<td>java.sql.Time</td>
</tr>
</tbody>
</table>
Table 1. InfoSphere DataStage data types and their equivalent Java data types (continued)

<table>
<thead>
<tr>
<th>InfoSphere DataStage data types</th>
<th>Java data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>java.sql.Timestamp</td>
</tr>
<tr>
<td>TinyInt</td>
<td>java.lang.Short</td>
</tr>
<tr>
<td>VarBinary</td>
<td>byte[]</td>
</tr>
<tr>
<td>VarChar</td>
<td>java.lang.String</td>
</tr>
</tbody>
</table>

Data type conversions from Java to DataStage data types (Java Integration stage)

In an output link where DataStage columns are set from the Java types produced by the Java Integration stage, the Java Integration stage converts Java data types to InfoSphere DataStage data types.

Likewise, after metadata is imported through the Java Integration stage, the Java data types are converted to InfoSphere DataStage data types. The following table shows the mapping rules between Java data types and InfoSphere DataStage data types.

Table 2. Java data types and their equivalent InfoSphere DataStage data types

<table>
<thead>
<tr>
<th>Java data type</th>
<th>InfoSphere DataStage data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean/java.lang.Boolean</td>
<td>Bit</td>
</tr>
<tr>
<td>Note: only applicable for Java Bean or UDF case</td>
<td></td>
</tr>
<tr>
<td>short/java.lang.Short</td>
<td>TinyInt</td>
</tr>
<tr>
<td>int/java.lang.Integer</td>
<td>SmallInt or Bit</td>
</tr>
<tr>
<td>long/java.lang.Long</td>
<td>Integer</td>
</tr>
<tr>
<td>java.math.BigInteger</td>
<td>BigInt</td>
</tr>
<tr>
<td>float/java.lang.Float</td>
<td>Real or Float</td>
</tr>
<tr>
<td>double/java.lang.Double</td>
<td>Double</td>
</tr>
<tr>
<td>byte[]</td>
<td>Binary or VarBinary or LongVarBinary</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>Char or VarChar or LongVarChar or NChar or NVarChar or LongNVarChar</td>
</tr>
<tr>
<td>java.sql.Date</td>
<td>Date</td>
</tr>
<tr>
<td>java.sql.Time</td>
<td>Time</td>
</tr>
<tr>
<td>java.sql.Timestamp</td>
<td>TimeStamp</td>
</tr>
<tr>
<td>java.math.BigDecimal</td>
<td>Decimal or Numeric</td>
</tr>
</tbody>
</table>

Time with microsecond resolution (Java Integration stage)

Time data type of DataStage Parallel Engine can have microsecond resolution, but the java.sql.Time has only millisecond resolution. As a result the last 3 digits of microseconds are truncated. The truncation also occurs when propagating columns from an input link to an output link. To avoid the truncation, the Java Integration stage API provides a class com.ibm.is.cc.javastage.api.TimeMicroseconds that preserves the time value to the microseconds specification.

The com.ibm.is.cc.javastage.api.TimeMicroseconds class inherits java.sql.Time and has microsecond resolution as well as hour, minute, and second. The
microseconds are not truncated when propagating columns from an input link to an output link. If you want to write the data with microseconds resolution to the output link, use this class.

For more information on the TimeMicroseconds class, see the Javadoc information for the Java Integration stage API.

## Retrieving Column Metadata on the Link (Java Integration stage)

The ColumnMetadata interface defines methods that are used to get the metadata such as column name, data type, and length associated to each column on the link. The column metadata that are associated with the Information Server DataStage® link can be retrieved by invoking the Link.getColumnMetadata(), the Link.getColumn(int) and the Link.getColumn(String) method.

### Methods provided by ColumnMetadata interface

- `getDataElementName()`
- `getDerivation()`
- `getDescription()`
- `getDisplaySize()`
- `getIndex()`
- `getName()`
- `getPrecision()`
- `setScale()`
- `getSQLType()`
- `getSQLTypeName()`
- `getType()`
- `hasMicrosecondResolution()`
- `isKey()`
- `isNull()`
- `isKey()`
- `isSigned()`
- `isUnicode()`

The following example retrieves the name and SQL type of the first column on the link.

```java
ColumnMetadata columnMetadata = m_inputLink.getColumn(0);
String columnName = columnMetadata.getName();
int sqlType = columnMetadata.getSQLType();
```

## Using user-defined properties (Java Integration stage)

You can use your Java code to define custom properties and use these property values in your Java code.

At job design time, the Java Integration stage editor calls the `getUserPropertyDefinitions()` method in your `Processor` class to get a list of user-defined property definitions and then show the properties in the editor panel to allow the users to specify the string value for each property.

The following example shows the sample implementation of the `getUserPropertyDefinitions()` method.
public List<propertyDefinition> getUserPropertyDefinitions()
{
    List<PropertyDefinition> list = new ArrayList<PropertyDefinition>();
    propList.add(new PropertyDefinition
    ("NumOfRecords", "10",
     "Number of Records",
     "Specifies the number of record to be generated.",
     PropertyDefinition.Scope.STAGE));
    propList.add(new PropertyDefinition
    ("WaveCount", "5",
     "Wave Count",
     "Specifies the number of record to be processed
      before writing end-of-wave marker.",
     PropertyDefinition.Scope.OUTPUT_LINK_ONLY));
    return list;
}

The getUserProperties() method of the Configuration interface returns a set of user-defined properties which forms as key=value pair. The getUserProperties() method of the Link interface returns a set of user-defined link properties which forms as key=value pair.

The following code gets the value of the NumOfRecords property which is set as stage properties, and the WaveCount property which is set as link properties.

public boolean validateConfiguration(
    Configuration configuration, boolean isRuntime) throws Exception
{
    // Specify current link configurations.
    m_outputLink = configuration.getOutputLink(0);

    Properties userStageProperties = configuration.getUserProperties();
    String propValue;
    // Fetch the value of "NumOfRecords" user property.
    // If it is not specified, use default value 10.
    // The minimum number of NumOfRecords is 0.
    // The maximum number of NumOfRecords is 100.
    // If specified value is out of range, return error.
    propValue = userStageProperties.getProperty("NumOfRecords");
    if (propValue != null)
    {
        m_numOfRecords = Integer.valueOf(propValue);
    }
    if (m_numOfRecords < 0 || m_numOfRecords > 100)
    {
        m_errors.add("Please set the NumOfRecords value between 1 to 100.");
    }

    // Fetch the value of "WaveCount" user property.
    // If it is not specified, use default value 5.
    // The minimum number of WaveCount is 0.
    // The maximum number of WaveCount is 100.
    // If specified value is out of range, return error.
    Properties userLinkProperties = m_outputLink.getUserProperties();
    propValue = userLinkProperties.getProperty("WaveCount");
    if (propValue != null)
    {
        m_waveCount = Integer.valueOf(propValue);
    }
    if (m_waveCount < 0 || m_waveCount > 100)
    {
        m_errors.add("Please set the waveCount value between 1 to 100.");
    }
}
Runtime column propagation (Java Integration stage)

InfoSphere DataStage allows you to define part of your schema and specify that if your job encounters extra columns that are not defined in the metadata when it actually runs, it will adopt these extra columns and propagate them through the rest of the job. This is known as runtime column propagation (RCP).

You can enable runtime column propagation for a project and set for individual links in the Output Page Columns tab. To enable runtime column propagation, select the Runtime column propagation check box.

Adding extra columns to output link

When runtime column propagation is enabled on a output link, your user code can add extra columns to this output link. The extra columns can be specified by Processor.getAdditionalOutputColumns() method, which is called for each output link whose RCP is enabled.

Following example shows how to add columns named "charCol" and "intCol" to a given output link.

```java
public List<ColumnMetadata> getAdditionalOutputColumns(Link outputLink, List<Link> inputLinks, Properties stageProperties) {
    List<ColumnMetadata> additionalColumns = new ArrayList<ColumnMetadata>();
    ColumnMetadata charCol = new ColumnMetadataImpl("charCol", ColumnMetadata.SQL_TYPE_CHAR);
    additionalColumns.add(charCol);
    ColumnMetadata intCol = new ColumnMetadataImpl("intCol", ColumnMetadata.SQL_TYPE_INTEGER);
    additionalColumns.add(intCol);
    return additionalColumns;
}
```

If you want to add columns that are in input link 0, but not in the output link, you can use helper method named the subtractColumnList() of the InputLink interface like below.

```java
public List<ColumnMetadata> getAdditionalOutputColumns(Link outputLink, List<Link> inputLinks, Properties stageProperties) {
    return inputLinks.get(0).subtractColumnList(outputLink);
}
```

Alternatively, you can use JavaBean to specify column definitions on the output links at runtime. If your user code uses JavaBean on the output links whose RCP is enabled, and both the JavaBean class and the Column mapping property is empty, the Java Integration stage automatically creates columns from the bean properties in a given JavaBean class, and add them to an output link.

The following table summarizes the SQL type created by the Java Integration stage. You can overwrite SQL type of columns by manually adding the corresponding column definition on the output link.

<table>
<thead>
<tr>
<th>Java type</th>
<th>SQL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte[]</td>
<td>Binary</td>
</tr>
<tr>
<td>boolean/java.lang.Boolean</td>
<td>Bit</td>
</tr>
<tr>
<td>Java type</td>
<td>SQL type</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>short/java.lang.Short</td>
<td>TinyInt</td>
</tr>
<tr>
<td>int/java.lang.Integer</td>
<td>SmallInt</td>
</tr>
<tr>
<td>double/java.lang.Double</td>
<td>Double</td>
</tr>
<tr>
<td>float/java.lang.Float</td>
<td>Float</td>
</tr>
<tr>
<td>long/java.lang.Long</td>
<td>Integer</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>VarChar</td>
</tr>
<tr>
<td>java.math.BigInteger</td>
<td>BigInt</td>
</tr>
<tr>
<td>java.math.BigDecimal</td>
<td>Decimal</td>
</tr>
<tr>
<td>java.sql.Date</td>
<td>Date</td>
</tr>
<tr>
<td>java.sql.Time</td>
<td>Time</td>
</tr>
<tr>
<td>com.ibm.is.cc.javastage.api.TimeMicroseconds.class</td>
<td>Time</td>
</tr>
<tr>
<td>java.sql.Timestamp</td>
<td>Timestamp</td>
</tr>
</tbody>
</table>

If the stage has a single input link and one or more output links, the Java Integration stage automatically adds all columns on the input link that are not present in the output link whose RCP is enabled.

Transferring the column data from input to output

Java Integration stage API provides the functionality to query column metadata dynamically at runtime, and access the data. You need to create code to read the data of the propagated columns on the input link, and write them to the output link for which the RCP is enabled.

Java Integration stage also provides the functionality to automatically transfer the column data from an input link to output link if the stage has a single input link and one or more output links. In this case, your user code is not required to transfer the data of the propagated columns on the input link. For more information, see the com.ibm.is.cc.javastage.api.ColumnTransferBehavior class in the Javadoc information for the Java Integration stage API.

Logging messages with the Java Integration stage

You can use the Logger class (com.ibm.is.cc.javastage.api.Logger) to log both runtime and design-time messages for your job.

At runtime, the messages are written to the job log. For example, your user code can load informational messages by calling Logger.information(). The runtime code of the Java Integration stage connector also logs messages to the job log.

The design-time messages are stored in the Connector Access Service log that you can view in the InfoSphere Information Server Web Console. For example, your user code can debug messages by calling Logger.debug(). The design-time code of the Java Integration stage also logs messages to the Connector Access Service log.

For more information on the Logger class, see the Javadoc information for the Java Integration stage API. Also see Chapter 6, “Troubleshooting (Java Integration stage),” on page 43.
Using JavaBeans (Java Integration stage)

The Java Integration stage API supports JavaBeans that allow the Java Integration stage to access existing Java code.

The Java Integration stage assumes the following JavaBeans conventions:
- The class must have a public default constructor (no-argument).
- The class must have getter and setter for each property.

The following types of JavaBeans property are supported by the Java Integration stage:
- Boolean/java.lang.Boolean
- byte[]
- short/java.lang.Short
- int/java.lang.Integer
- long/java.lang.Long
- float/java.lang.Float
- double/java.lang.Double
- java.lang.String
- java.math.BigInteger
- java.math.BigDecimal
- java.sql.Time
- java.sql.Timestamp
- java.sql.Date

The following example shows how to use the `samples.InputBean` class for an input link and the `samples.OutputBean` class for an output link.

`JavaBeansTransformer.java`

```java
package samples;

import com.ibm.is.cc.javastage.api.*;

public class JavaBeansTransformer extends Processor {
    private InputLink m_inputLink;
    private OutputLink m_outputLink;
    private OutputLink m_rejectLink;

    public Capabilities getCapabilities() {
        Capabilities capabilities = new Capabilities();
        // Set minimum number of input links to 1
        capabilities.setMinimumInputLinkCount(1);
        // Set maximum number of input links to 1
        capabilities.setMaximumInputLinkCount(1);
        // Set minimum number of output stream links to 1
        capabilities.setMinimumOutputStreamLinkCount(1);
        // Set maximum number of output stream links to 1
        capabilities.setMaximumOutputStreamLinkCount(1);
        // Set maximum number of reject links to 1
        capabilities.setMaximumRejectLinkCount(1);
        // Set is Wave Generator to false
        capabilities.setIsWaveGenerator(false);
        return capabilities;
    }
```
public boolean validateConfiguration(
    Configuration configuration, boolean isRuntime)
throws Exception {
    // Specify current link configurations.
    m_inputLink = configuration.getInputLink(0);
    m_outputLink = configuration.getOutputLink(0);
    if (configuration.getRejectLinkCount() == 1) {
        m_rejectLink = m_inputLink.getAssociatedRejectLink();
    }
    return true;
}

public void process() throws Exception {
    OutputRecord outputRecord = m_outputLink.getOutputRecord();

    // Loop until there is no more input data
    do {
        InputRecord record = m_inputLink.readRecord();
        if (record == null) {
            // End of data
            break;
        }

        // Get the object from the input row.
        InputBean inputBean = (InputBean) record.getObject();

        // Get the value from name column of the input link.
        // If the value contains "*" character, mark reject flag
        // and send the record
        // to reject link in later processing.
        boolean fReject = false;
        String name = inputBean.getFirstName();
        if ((name == null) || (name.indexOf('*') >= 0)) {
            fReject = true;
        }

        if (!fReject) {
            // Send record to output
            OutputBean outputBean = new OutputBean();
            outputBean.setEmpno(inputBean.getEmpno());
            outputBean.setFirstName(inputBean.getFirstName().toUpperCase());
            outputBean.setLastName(inputBean.getLastName().toUpperCase());
            outputBean.setHireDate(inputBean.getHireDate());
            outputBean.setEdLevel(inputBean.getEdLevel());
            outputBean.setSalary(inputBean.getSalary());
            outputBean.setBonus(inputBean.getBonus());
            outputBean.setLastUpdate(inputBean.getLastUpdate());
            outputRecord.putObject(outputBean);
            m_outputLink.writeRecord(outputRecord);
        } else if (m_rejectLink != null) {
            // Reject record. This transfers the row to the reject link.
            // The same kind of forwarding is also possible for regular stream
            // links.
            RejectRecord rejectRecord = m_rejectLink.getRejectRecord(record);

            // Reject record can contain additional columns "ERRORTEXT" and "ERRORCODE".
        }
    } while (true);
}

Chapter 2. Writing Java code to use in jobs
The field will be shown as columns in rejected output records.
rejectRecord.setErrorText("Name field contains *");
rejectRecord.setErrorCode(123);
m_rejectLink.writeRecord(rejectRecord);
}
while (true);
}
}
public Class<InputBean> getBeanForInput(Link inputLink)
{
    return InputBean.class;
}
public Class<OutputBean> getBeanForOutput(Link outputLink)
{
    return OutputBean.class;
}

**InputBean.java**

```java
package samples;
import java.sql.Date;
import java.sql.Time;
public class InputBean {
    private long m_empno;
    private String m_firstname;
    private String m_lastname;
    private Date m_hiredate;
    private int m_edlevel;
    private Double m_salary;
    private double m_bonus;
    private Time m_lastupdate;

    /**
     * Fetches the value of the empno field.
     * @return long value of empno field
     */
    public long getEmpno() {
        return m_empno;
    }

    /**
     * Set the value of the empno field.
     * @param empno value of the empno field.
     */
    public void setEmpno(long empno) {
        m_empno = empno;
    }

    /**
     * Fetches the value of the firstname field.
     * @return String value of firstname field
     */
    public String getFirstName() {
        return m_firstname;
    }
```
/**
 * Set the value of the firstname field.
 * @param firstname value of the firstname field.
 */
public void setFirstName(String firstname)
{
    m_firstname = firstname;
}

/**
 * Fetches the value of the lastname field.
 * @return String value of lastname field
 */
public String getLastName()
{
    return m_lastname;
}

/**
 * Set the value of the lastname field.
 * @param lastname value of the lastname field.
 */
public void setLastName(String lastname)
{
    m_lastname = lastname;
}

/**
 * Fetches the value of the hiredate field.
 * @return Date value of hiredate field
 */
public Date getHireDate()
{
    return m_hiredate;
}

/**
 * Set the value of the hiredate field.
 * @param hiredate value of the hiredate field.
 */
public void setHireDate(Date hiredate)
{
    m_hiredate = hiredate;
}

/**
 * Fetches the value of the edlevel field.
 * @return int value of edlevel field
 */
public int getEdLevel()
{
    return m_edlevel;
}

/**
 * Set the value of the edlevel field.
 * @param edlevel value of the edlevel field.
 */
public void setEdLevel(int edlevel)
m_edlevel = edlevel;
}

/**
 * Fetches the value of the salary field.
 * @return Double value of salary field
 */
public Double getSalary()
{
    return m_salary;
}

/**
 * Set the value of the salary field.
 * @param salary value of the salary field.
 */
public void setSalary(Double salary)
{
    m_salary = salary;
}

/**
 * Fetches the value of the bonus field.
 * @return double value of bonus field
 */
public double getBonus()
{
    return m_bonus;
}

/**
 * Set the value of the bonus field.
 * @param bonus value of the bonus field.
 */
public void setBonus(double bonus)
{
    m_bonus = bonus;
}

/**
 * Fetches the value of the lastupdate field.
 * @return Time value of lastupdate field
 */
public Time getLastUpdate()
{
    return m_lastupdate;
}

/**
 * Set the value of the lastupdate field.
 * @param lastupdate value of the lastupdate field.
 */
public void setLastUpdate(Time lastupdate)
{
    m_lastupdate = lastupdate;
}

OutputBean.java
package samples;
import java.sql.Date;
import java.sql.Time;

public class OutputBean
{
    private long m_empno;
    private String m_firstname;
    private String m_lastname;
    private Date m_hiredate;
    private int m_edlevel;
    private Double m_salary;
    private double m_bonus;
    private double m_income;
    private Time m_lastupdate;

    /**
     * Fetches the value of the empno field.
     * @return long value of empno field
     */
    public long getEmpno()
    {
        return m_empno;
    }

    /**
     * Set the value of the empno field.
     * @param empno value of the empno field.
     */
    public void setEmpno(long empno)
    {
        m_empno = empno;
    }

    /**
     * Fetches the value of the firstname field.
     * @return String value of firstname field
     */
    public String getFirstName()
    {
        return m_firstname;
    }

    /**
     * Set the value of the firstname field.
     * @param firstname value of the firstname field.
     */
    public void setFirstName(String firstname)
    {
        m_firstname = firstname;
    }

    /**
     * Fetches the value of the lastname field.
     * @return String value of lastname field
     */
    public String getLastName()
    {
        return m_lastname;
    }
}

Chapter 2. Writing Java code to use in jobs 23
/**
 * Set the value of the lastname field.
 * @param lastname value of the lastname field.
 */
public void setLastName(String lastname)
{
    m_lastname = lastname;
}

/**
 * Fetches the value of the hiredate field.
 * @return Date value of hiredate field
 */
public Date getHireDate()
{
    return m_hiredate;
}

/**
 * Set the value of the hiredate field.
 * @param hiredate value of the hiredate field.
 */
public void setHireDate(Date hiredate)
{
    m_hiredate = hiredate;
}

/**
 * Fetches the value of the edlevel field.
 * @return int value of edlevel field
 */
public int getEdLevel()
{
    return m_edlevel;
}

/**
 * Set the value of the edlevel field.
 * @param edlevel value of the edlevel field.
 */
public void setEdLevel(int edlevel)
{
    m_edlevel = edlevel;
}

/**
 * Fetches the value of the salary field.
 * @return Double value of salary field
 */
public Double getSalary()
{
    return m_salary;
}

/**
 * Set the value of the salary field.
 * @param salary value of the salary field.
 */
public void setSalary(Double salary)
{
m_salary = salary;
}

/**
 * Fetches the value of the bonus field.
 * @return double value of bonus field
 */
public double getBonus()
{
    return m_bonus;
}

/**
 * Set the value of the bonus field.
 * @param bonus value of the bonus field.
 */
public void setBonus(double bonus)
{
    m_bonus = bonus;
}

/**
 * Fetches the value of the lastupdate field.
 * @return Time value of lastupdate field
 */
public Time getLastUpdate()
{
    return m_lastupdate;
}

/**
 * Set the value of the lastupdate field.
 * @param lastupdate value of the lastupdate field.
 */
public void setLastUpdate(Time lastupdate)
{
    m_lastupdate = lastupdate;
}

If your Java code uses JavaBeans as representations of records on the link, you must override the getBeanForInput() and the getBeanforOutput() methods in the Processor class. Your Java code must return the java.lang.Class of JavaBeans class corresponding to each link. The Java Integration stage will invoke these method at the time of initialization.

public Class<InputBean> getBeanForInput(Link inputLink)
{
    return InputBean.class;
}

public Class<OutputBean> getBeanForOutput(Link outputLink)
{
    return OutputBean.class;
}

The JavaBeans class corresponding to an input link can be instantiated by the Java Integration stage, and your Java code can get this instance by calling the getObject() method of the InputLink interface.
InputBean inputBean = (InputBean) record.getObject();

String name = inputBean.getFirstName();

Before your Java code writes a record to an output link which is associated with JavaBeans, your Java code must instantiate a JavaBeans object for the output link, and set the value for each bean properties. The type of JavaBeans object needs to match the type specified by the getBeanForOutput() method.

// Send record to output
OutputBean outputBean = new OutputBean();
outputBean.setEmpno(inputBean.getEmpno());
outputBean.setFirstName(inputBean.getFirstName().toUpperCase());
outputBean.setLastName(inputBean.getLastName().toUpperCase());
outputBean.setHireDate(inputBean.getHireDate());
outputBean.setEdLevel(inputBean.getEdLevel());
outputBean.setSalary(inputBean.getSalary());
outputBean.setBonus(inputBean.getBonus());
outputBean.setLastUpdate(inputBean.getLastUpdate());

Finally, your code must call the putObject(Object) method of the OutputRecord interface to set this JavaBeans object to the output record.

outputRecord.putObject(outputBean);

User Defined Function (UDF) - Java Integration stage

The Java Integration stage supports the execution of existing user-defined functions that use JavaBeans or primitive types (that are supported by Java Stage) in their calling interface.

To execute the existing user-defined functions, the number of parameters must match the number of input links, and the return value bean must map to an output link. An exception to this rule is that when the Java Integration stage is used as a target, it does not matter whether the user-defined function returns a value or not.

The following example code shows a user-defined function that will combine two input records, and write the result to an output link:

```java
public class UserDefinedFunction
{
  /**
   * Passes primitive type double and a bean as UDF arguments and
   * returns a bean.
   * @param commission commission
   * @param input (InputBean) object.
   * @return output (UDFOutputBean) object.
   */
  public UDFOutputBean AnnualIncome(double commission, InputBean input)
  {UDFOutputBean output = new UDFOutputBean();

    output.setEmpno(input.getEmpno());
    output.setFirstName(input.getFirstName().toUpperCase());
    output.setLastName(input.getLastName().toUpperCase());

    double total = commission +
    input.getSalary().doubleValue() +
    input.getBonus();
  }
}
```
output.setIncome(total);
    return output;
}
Chapter 3. Designing jobs (Java Integration stage)

You can use the Java Integration stage to integrate Java code and develop jobs that read, write, and load data.

Procedure
1. Add a Java Integration stage to a job
   a. Optional: Migrate a legacy Java Pack job to Java Integration stage
2. To set up the Java Integration stage as a source to retrieve data from Java code:
   a. Configure Java Integration stage as a source
   b. Set up column definitions
   c. Map link index and link names
3. To set up the Java Integration stage as a target to pass data to Java code:
   a. Configure the Java Integration stage as a target
   b. Set up column definitions
   c. Map link index and link names
4. To set up the Java Integration stage as a transformer to transform data on input link, and write to output link:
   a. Configure the Java Integration stage as a transformer
   b. Set up column definitions
   c. Map link index and link names
5. Look up data by using reference links
6. Compile and run the job

Adding a Java Integration stage to a job

Use the InfoSphere DataStage and QualityStage® Designer client to add a Java Integration stage to a job.

Procedure
1. From the Designer client, select File > New from the menu.
2. In the New window, select the Parallel Job icon, and click OK.
3. On the left side of the Designer client in the Palette menu, select the Real Time category.
4. Locate Java Integration in the list.
5. Drag the Java Integration stage icon to the job design canvas.
6. Connect input and output links to the Java Integration stage.
7. Double click the Java Integration stage icon and select the Stage tab to enter or modify the following attributes:
   - Stage name: Modify the default name of the Stage. You can enter up to 255 characters. Alternatively, you can modify the name of the stage in the job design canvas.
   - Description: Enter an optional description of the stage.
8. Click Save.
What to do next

Define properties for the Java Integration stage.

Retrieving data from Java code (Java Integration stage)

To read data from your Java code by using the Java Integration stage, you need to integrate a Java code supported by the Java Integration stage and configure the Java Integration stage to process data as a source. As a source, the connector extracts or reads data from your Java code.

The following figure shows an example of using the Java Integration stage to read data. In this case, the Java Integration stage `Java_Stage_0` reads data and writes it to the files Peek_1 and Peek_2. When you configure the Java Integration stage to read data, you can create 1 or more output links Peek_1 and Peek_2, which is shown in the figure below transferring rows from `Java_Stage_0` to Peek_1 and Peek_2.

![Figure 1. Example of reading data](image)

Configuring Java Integration stage as a source

You can configure the Java Integration stage to process data as a source for 1 or more output links.

**Procedure**

1. On the job design canvas, double-click the Java Integration stage icon.
2. Select the Output tab and select the output link that you want to edit from the Output name (downstream stage) drop down list. By editing the output link you are setting up Java Integration stage to be the source.
3. Specify an optional description of the output link in the General tab.
4. Optional: Click Configure to configure additional properties. Depending on your user code the Custom Property Editor, Column Mapping Editor or the Column Metadata Importer window is displayed.
   a. You need to specify values for the properties. If your user code exposes user-defined properties by using the
Processor.getUserPropertyDefinitions() method, the **Custom Property Editor** window is displayed where you can specify the value of each property.

b. If your user code uses JavaBeans in its interface (by implementing the Processor.getBeanForInput() and Processor.getBeanForOutput() methods), the **Column Mapping Editor** window is displayed. Populate DataStage column schema based on the JavaBeans properties and then map JavaBeans properties to the DataStage columns. If you create a job where the links contain no columns, then the initial **Column Mapping Editor** contains empty tables. Click **Browse Objects** to launch **Select Bean Properties** window, and select the bean properties or user-defined function (UDF) arguments to be imported in the **Column Mapping Editor**, and then click OK. Click **Finish**. You can also map JavaBean properties to existing DataStage columns, instead of creating new columns.

c. If your user code implements the Processor.getColumnMetadataForInput() and Processor.getColumnMetadataForOutput() methods instead of implementing the Processor.getBeanForInput() and Processor.getBeanForOutput() methods, the **Column Metadata Importer** window is displayed. In the **Column Metadata Importer** window select the column metadata to populate DataStage column schema from a list of ColumnMetadata instances that are returned by the Processor.getColumnMetadataForInput() and Processor.getColumnMetadataForOutput() methods for each link. Click **Browse Objects** and select the column metadata to be populated in the job. Click **Finish**.

5. Specify required details in the **Properties** tab and the **Advanced** tab.

6. Click **OK** to save the connection settings that you specified.

### Setting up column definitions (Java Integration stage)

You can set up column definitions for a link and also customize the columns grid, save column definitions for later use, and load predefined column definitions from the repository.

**Procedure**

1. On the job design canvas, double-click the **Java Integration stage** icon.

2. Select the **Output** tab and select the output link that you want to edit from the **Output name (downstream stage)** drop down list.

3. On the **Columns** tab, modify the columns grid to specify the metadata that you want to define.
   
   a. Right-click within the grid, and select **Properties** from the menu.

   b. In the Grid properties window, select the properties that you want to display and the order that you want them to be displayed. Then, click **OK**.

4. Enter column definitions for the table by using one of the following methods:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method 1</td>
<td>1. In the Column name column, double-click inside the appropriate cell and type a column name.</td>
</tr>
<tr>
<td></td>
<td>2. For each cell in the row, double-click inside the cell and select the options that you want.</td>
</tr>
<tr>
<td></td>
<td>3. In the Description column, double-click inside the appropriate cell and type a description.</td>
</tr>
<tr>
<td>Method 2</td>
<td>1. Right-click within the grid, and select Edit row from the menu.</td>
</tr>
<tr>
<td></td>
<td>2. In the Edit column metadata window, enter the column metadata.</td>
</tr>
</tbody>
</table>

5. To share metadata between several columns, select the columns that you want to share metadata.
   a. Right-click and select Propagate values.
   b. In the Propagate column values window, select the properties that you want the selected columns to share.

6. To save the column definitions as a table definition in the repository, click Save.
   a. Enter the appropriate information in the Save Table Definition window, and then click OK.
   b. In the Save Table Definition As window, select the folder where you want to save the table definition, and then click Save.

7. To load column definitions from the repository, click Load.
   a. In the Table Definitions window, select the table definition that you want to load, and then click OK.
   b. In the Select Columns window, use the arrow buttons to move columns from the Available columns list to the Selected columns list. Click OK.

**Associating link indices with links (Java Integration Stage)**
You can associate link indices with links when there are multiple output links.

**Procedure**
1. On the job design canvas, double-click the output link icon (that connects to the Java Integration Stage icon) for which you want to change the order.
2. Select the Link Ordering tab. This tab allows you to specify how output links correspond to numeric link labels. The numeric link label corresponds to the index of the link accessed through the Java Integration Stage API. To rearrange the links, choose an output link and click the up arrow button or the down arrow button.
3. Click OK to save the mapping details.

**Passing data to Java code (Java Integration stage)**
To pass data to your Java code by using the Java Integration stage, you need to integrate a Java code that is supported by the Java Integration stage and configure the Java Integration stage to process data as a target.

The following figure shows an example of using the Java Integration stage to pass data to Java code. In this case, the row generator reads data from the files.
Row_Generator_5 and Row_Generator_7 and then the Java Integration stage Java_Stage_0 inserts, updates, or deletes data as required.

**Configuring Java Integration stage as a target**

You can configure Java Integration stage as a target to write data. Java Integration stage as a target can have 1 or more input links, and 0 or more reject links.

**Procedure**

1. On the job design canvas, double-click the Java Integration stage icon.
2. Select the Input tab and select the input link that you want to edit from the Input name (upstream stage) drop down list. By editing the input link you are setting up Java Integration stage to be the target.
3. Specify an optional description of the input link in the General tab.
4. Optional: Click Configure to configure additional properties. Depending on the user code the Custom Property Editor, Column Mapping Editor or the Column Metadata Importer window is displayed.
   a. You need to specify values for the properties. If your user code exposes user-defined properties by using the Processor.getUserPropertyDefinitions() method, the Custom Property Editor window is displayed where you can specify the value of each property.
   b. If your user code uses JavaBeans in its interface (by implementing the Processor.getBeanForInput() and Processor.getBeanForOutput() methods), the Column Mapping Editor window is displayed. Populate DataStage column schema based on the JavaBeans properties and then map JavaBeans properties to the DataStage columns. If you create a job where the links contain no columns, then the initial Column Mapping Editor contains empty tables. Click Browse Objects to launch Select Bean Properties window, and select the bean properties or user-defined function (UDF) arguments to be imported in the Column Mapping Editor, and then click OK. Click Finish. You can also map JavaBean properties to existing DataStage columns, instead of creating new columns.
c. If your user code implements the `Processor.getColumnMetadataForInput()` and `Processor.getColumnMetadataForOutput()` methods instead of implementing the `Processor.getBeanForInput()` and `Processor.getBeanForOutput()` methods, the **Column Metadata Importer** window is displayed. In the **Column Metadata Importer** window select the column metadata to populate DataStage column schema from a list of `ColumnMetadata` instances that are returned by the `Processor.getColumnMetadataForInput()` and `Processor.getColumnMetadataForOutput()` methods for each link. Click **Browse Objects** and select the column metadata to be populated in the job. Click **Finish**.

5. Specify required details in the **Properties** tab and the **Advanced** tab.
6. Click **OK** to save the settings that you specified.

### Setting up column definitions (Java Integration stage)

You can set up column definitions for a link and also customize the columns grid, save column definitions for later use, and load predefined column definitions from the repository.

#### Procedure

1. On the job design canvas, double-click the **Java Integration stage** icon.
2. Select the **Input** tab and select the input link that you want to edit from the **Input name (upstream stage)** drop down list.
3. On the **Columns** tab, modify the columns grid to specify the metadata that you want to define.
   a. Right-click within the grid, and select **Properties** from the menu.
   b. In the Grid properties window, select the properties that you want to display and the order that you want them to be displayed. Then, click **OK**.
4. Enter column definitions for the table by using one of the following methods:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Method 1** | 1. In the **Column name** column, double-click inside the appropriate cell and type a column name.  
2. For each cell in the row, double-click inside the cell and select the options that you want.  
3. In the **Description** column, double-click inside the appropriate cell and type a description. |
| **Method 2** | 1. Right-click within the grid, and select **Edit row** from the menu.  
2. In the Edit column metadata window, enter the column metadata. |

5. To share metadata between several columns, select the columns that you want to share metadata.
   a. Right-click and select **Propagate values**.
   b. In the Propagate column values window, select the properties that you want the selected columns to share.
6. To save the column definitions as a table definition in the repository, click **Save**.
a. Enter the appropriate information in the Save Table Definition window, and then click OK.

b. In the Save Table Definition As window, select the folder where you want to save the table definition, and then click Save.

7. To load column definitions from the repository, click Load.
   a. In the Table Definitions window, select the table definition that you want to load, and then click OK.
   b. In the Select Columns window, use the arrow buttons to move columns from the Available columns list to the Selected columns list. Click OK.

**Associating link indices with links (Java Integration Stage)**
You can associate link indices with links when there are multiple input links.

**Procedure**
1. On the job design canvas, double-click the input link icon (that connects to the Java Integration Stage icon) for which you want to change the order.
2. Select the Link Ordering tab. This tab allows you to specify how output links correspond to numeric link labels. The numeric link label corresponds to the index of the link accessed through the Java Integration Stage API. To rearrange the links, choose an output link and click the up arrow button or the down arrow button.
3. Click OK to save the mapping details.

**Transforming data (Java Integration stage)**
To transform data by using the Java Integration stage, you need to integrate a Java code that is supported by the Java Integration stage and configure the Java Integration stage to transform data on input link, and write results to output link.

The following figure shows an example of using the Java Integration stage as a transformer. In this case, the row generator reads data from Row_Generator_5 and then the Java Integration stage Java_Stage_0 writes the transformed data to Peek_1 and to the reject link Peek_2.
Configuring Java Integration stage as a transformer

You can configure Java Integration stage as a transformer to transform external data. Java Integration stage as a transformer can have 1 or more input links, and 1 or more output or reject links.

Procedure

1. On the job design canvas, double-click the Java Integration stage icon.
2. Select the Input tab and select the input link that you want to edit from the Input name (upstream stage) drop down list or select the Output tab and select the output link that you want to edit from the Output name (downstream stage) drop down list. By editing the input link and output link, you are setting up Java Integration stage to be the transformer.
3. Specify an optional description of the input link or the output link in the General tab.
4. Optional: Click Configure to configure additional properties. Depending on the user code the Custom Property Editor, Column Mapping Editor or the Column Metadata Importer window is displayed.
   a. You need to specify values for the properties. If your user code exposes user-defined properties by using the Processor.getUserPropertyDefinitions() method, the Custom Property Editor window is displayed where you can specify the value of each property.
   b. If your user code uses JavaBeans in its interface (by implementing the Processor.getBeanForInput() and Processor.getBeanForOutput() methods), the Column Mapping Editor window is displayed. Populate DataStage column schema based on the JavaBeans properties and then map JavaBeans properties to the DataStage columns. If you create a job where the links contain no columns, then the initial Column Mapping Editor contains empty tables. Click Browse Objects to launch Select Bean Properties window, and select the bean properties or user-defined function (UDF)
arguments to be imported in the Column Mapping Editor, and then click OK. Click Finish. You can also map JavaBean properties to existing DataStage columns, instead of creating new columns.

If your user code implements the `Processor.getColumnMetadataForInput()` and `Processor.getColumnMetadataForOutput()` methods instead of implementing the `Processor.getBeanForInput()` and `Processor.getBeanForOutput()` methods, the Column Metadata Importer window is displayed. In the Column Metadata Importer window select the column metadata to populate DataStage column schema from a list of `ColumnMetadata` instances that are returned by the `Processor.getColumnMetadataForInput()` and `Processor.getColumnMetadataForOutput()` methods for each link. Click Browse Objects and select the column metadata to be populated in the job. Click Finish.

5. Specify required details in the Properties tab and the Advanced tab.

6. Click OK to save the settings that you specified.

Setting up column definitions (Java Integration stage)

You can set up column definitions for a link and also customize the columns grid, save column definitions for later use, and load predefined column definitions from the repository.

Procedure

1. On the job design canvas, double-click the Java Integration stage icon.

2. Select the Input tab and select the input link that you want to edit from the Input name (upstream stage) drop down list or select the Output tab and select the output link that you want to edit from the Output name (downstream stage) drop down list.

3. On the Columns tab, modify the columns grid to specify the metadata that you want to define.
   a. Right-click within the grid, and select Properties from the menu.
   b. In the Grid properties window, select the properties that you want to display and the order that you want them to be displayed. Then, click OK.

4. Enter column definitions for the table by using one of the following methods:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| Method 1   | 1. In the Column name column, double-click inside the appropriate cell and type a column name.  
            2. For each cell in the row, double-click inside the cell and select the options that you want.  
            3. In the Description column, double-click inside the appropriate cell and type a description. |
| Method 2   | 1. Right-click within the grid, and select Edit row from the menu.  
            2. In the Edit column metadata window, enter the column metadata.                                                                                      |

5. To share metadata between several columns, select the columns that you want to share metadata.
   a. Right-click and select Propagate values.
b. In the Propagate column values window, select the properties that you want the selected columns to share.

6. To save the column definitions as a table definition in the repository, click **Save**.
   a. Enter the appropriate information in the Save Table Definition window, and then click **OK**.
   b. In the Save Table Definition As window, select the folder where you want to save the table definition, and then click **Save**.

7. To load column definitions from the repository, click **Load**.
   a. In the Table Definitions window, select the table definition that you want to load, and then click **OK**.
   b. In the Select Columns window, use the arrow buttons to move columns from the **Available columns** list to the **Selected columns** list. Click **OK**.

**Associating link indices with links (Java Integration stage)**
You can associate link indices with links when there are multiple input or output links.

**Procedure**
1. On the job design canvas, double-click the input or output link icon (that connects to the **Java Integration Stage** icon) for which you want to change the order.
2. Select the **Link Ordering** tab. This tab allows you to specify how the input or output links correspond to numeric link labels. The numeric link label corresponds to the index of the link accessed through the Java Integration Stage API. To rearrange the links, choose the required input or output link and click the up arrow button or the down arrow button.
3. Click **OK** to save the mapping details.

**Looking up data by using reference links (Java Integration stage)**
You can look up data by using a reference link to link the Java Integration stage to a Lookup stage. Java Integration stage only supports normal lookup. Java Integration stage does not support Sparse lookup.

**Procedure**
1. On the job design canvas, drag a **Java Integration** stage icon and a **Lookup stage** icon to the job design canvas. (The **Lookup stage** is located in the **Processing** category of the **Palette** menu.)
2. Join the stages by dragging a link from Java Integration stage to the Lookup stage.
3. Right-click the link, and select **Convert to Reference** from the menu. The line changes to a dashed line to show that the link is a reference link.
4. Click **OK**.
Chapter 4. Migrating the legacy Java Pack jobs to Java Integration stage

Use the Connector Migration Tool to view and migrate the legacy Java Pack jobs to Java Integration stage. The Java Integration stage is backward compatible to the current Java Pack so that the existing Java classes work unmodified with the Java Integration stage. The Java Integration stage also supports the existing API that is exposed in the existing Java Pack. The Java Pack API will not be extended beyond its current capability. The supported migration paths are Java Transformer (Parallel) to Java Integration stage (Parallel) and Java Client (Parallel) to Java Integration stage (Parallel).

Procedure
1. Choose Start > Programs > IBM InfoSphere Information Server > Connector Migration Tool.
2. Log in by specifying the host name where the Designer client runs and identifying the project where the Java Pack jobs are defined.
3. Select View > View all migratable jobs to display all of the jobs that are in the project and that can be migrated to use Java Integration stage. Jobs that do not contain any stages that can be migrated are excluded from the job list. An icon indicates the status of each job. A gray icon indicates that the job cannot be migrated. A gray icon with a question mark indicates that the job might be successfully migrated.
4. Perform the following steps to analyze jobs:
   a. Highlight the job in the job list.
   b. Expand the job in the job list to view the stages in the job.
   c. Select one or more jobs, and click Analyze.

After analysis, the color of the job, stage, or property icon indicates whether or not it can be migrated.

Green icon
A green icon indicates that the job, stage, or property can be migrated.

Red icon
A red icon indicates that the job or stage cannot be migrated.

Orange icon
A orange icon indicates that a job or stage can be partially migrated and that a property in a stage has no equivalent in Java Integration stage.

Gray icon
A gray icon indicates that the job or stage is not eligible for migration.

The Connector Migration Tool displays internal property names, rather than the names that the stages display. To view a table that contains the internal name and the corresponding display name for each property, from the IBM InfoSphere DataStage and QualityStage Designer client, open the Stage Types folder in the repository tree. Double-click the stage icon, and then click the Properties tab to view the stage properties.

5. Click Preferences and choose how to migrate the job:
• Click **Clone and migrate cloned job** to make a copy of the job and then migrate the copy. The original job remains intact.

• Click **Back up job and migrate original job** to make a copy of the job and then migrate the original job.

• Click **Migrate original job** to migrate the job without making a backup.

6. Select the jobs and stages to migrate, and then click **Migrate**.

The selected jobs are migrated to the Java Integration stage and are placed in the same folder as the original job. If logging is enabled, a log file that contains a report of the migration task is created. After a job is successfully migrated, a green checkmark displays beside the job name in the Jobs list to indicate that the job has been migrated.
Chapter 5. Compiling and running Java Integration stage jobs

You can compile the Java Integration stage jobs into executable scripts that you can schedule and run.

Procedure
1. In the InfoSphere DataStage and QualityStage Designer client, open the job that you want to compile.
2. Click the Compile button.
3. If the Compilation Status area shows errors, edit the job to resolve the errors. After resolving the errors, click the Re-compile button.
4. When the job compiles successfully, click the Run button, and specify the job run options:
   a. Enter the job parameters as required.
   b. Click the Validate button to verify that the job will run successfully without actually extracting, converting, or writing data.
   c. Click the Run button to extract, convert, or write data.
5. To view the results of validating or running a job:
   a. In the Designer client, select Tools > Run Director to open the Director client.
   b. In the Status column, verify that the job was validated or completed successfully.
   c. If the job or validation fails, select View > Log to identify any runtime problems.
6. If the job has runtime problems, fix the problems, recompile, validate (optional), and run the job until it completes successfully.
Chapter 6. Troubleshooting (Java Integration stage)

When using the Java Integration stage, you might encounter errors that can be fixed by troubleshooting, adjusting values for properties or configuration. The most common types of errors that you might encounter are GUI errors and Runtime errors.

GUI errors

You might encounter the following warning dialogs:

- When you click the **Configure** button on the **Java Integration stage property editor**, you might encounter the following warning dialog:
  
  Failed to instantiate the resource wrapper class

- When you click the **Select** button on the stage property **Usage - User class** on the **Java Integration stage property editor**, you might encounter the following warning dialog:
  
  Failed to send the request to the handler: The agent at HOST123:31531 is not available.

To correct the error: Verify that the ASB agent is working and restart the ASB agent if necessary.

Design time log helps understand the errors that occur in GUI. It is stored as **Connector Access Service log** that can be retrieved via the **Information Server Web Console**. The user’s code logs debug message by calling **Logger.debug()**. The design-time code of Java Integration stage connector also logs the message to **Connector Access Service log**.

Use the following steps to configure the log view for Java Integration stage and then view it.

To configure perform the following steps:

1. Log-in to Information Server Web Console.
2. In **Administration** tab, select **Log Management > Logging Components**.
3. Select **ConnectorAccess** and click on **Manage Configurations**.
4. Select **ConnectorAccess.WARN** and click **Copy**.
5. Change name to **ConnectorAccess.ALL**. Set **Threshold** to **All**. Change all severity drop downs to **All**.
6. Click **Save and Close**.

To view the log perform the following steps:

1. Select **ConnectorAccess.ALL** and click **Set as Active**. Wait 1 minute for changes to make their ways to clients and ASBAgent.
2. In **Administration** tab, select **Log Management > Log Views**.
3. Click **New**.
4. Enter a name (e.g. 'Java Integration stage') and description (e.g. 'Java Integration stage log view').
5. In **Severity Levels** click the **All** checkbox.
6. Expand **Categories** section and click on **Browse**. Select **ISF-CAS-HANDLER** and click on **OK**. Click **Save and Close**.
7. In Administration tab, select Log Management > Log Views,
8. Select the Java Integration stage log view created in Step 4 and click on View Log.

**Runtime errors**

After you configured and compiled your Java Integration stage job successfully, you might encounter the following runtime errors:

- **Stage property and configuration related issues**
  - You might encounter the following fatal error when running a Java Integration stage job:
    ```
    com.ascential.e2.common.CC_Exception: java.lang.ClassNotFoundException: userClass
    at java.net.URLClassLoader.findClass(URLClassLoader.java:434)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:653)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:619)
    at com.ibm.is.cc.javastage.connector.CC_JavaConnection.connect
      (CC_JavaConnection.java:152)
    at com.ibm.is.cc.javastage.connector.CC_JavaConnection.connect
      (CC_JavaConnection.java:247)
    ```
  
  This error might occur because a user class `userClass` could not be found in the classpath specified at the Java Integration stage property editor.

  - You might encounter the following fatal error when running a Java Integration stage job:
    ```
    Connection test failed (PXBridgeOp::negotiate, file pxbridge.c, line 1,684)
    ```
  
  This error might occur because a user class failed the connection test at the initialization time. You might have specified a wrong user class inherited from neither `com.ibm.is.cc.javastage.api.Processor` nor `com.ascentialsoftware.jds.Stage`.

  To correct the errors: You need to place the right user class file inherited from Processor or Stage class, or a jar file containing the class into a directory which DataStage engine can access, and set the path name at the stage property Usage - Java - Classpath on the Java Integration stage property editor. To confirm that the correct classpath was specified, click the Select button on the property Usage - User class and select the user class from the dialog panel.

- **JVM related issues**
  - You might encounter the following fatal error when running a job which has multiple Java Integration stages:
    ```
    JVMDUMP006I Processing dump event "systhrow", detail "java/lang/OutOfMemoryError"
    - please wait.
    JVMDUMP032I JVM requested Heap dump using 
      'C:\IBM\InformationServer\Server\Projects\UT\heapdump.20111117.115037.4060.0001.phd'
      in response to an event
    ```
  
  The conductor runs as a single process and creates a single JVM with JVM options specified at one of Java Integration stages in the job to perform the initialization for all the Java Integration stages. This error might occur because the JVM option maximum heap size might not be enough although it is enough for that Java Integration stage since the user classes for all the Java Integration stages are instantiated in that JVM.

  - You might encounter the following fatal error when running a job which has a Java Integration stage and a XML stage:
The XML stage is a Java based stage as well as Java Integration stage so that both stages are instantiated in a single JVM in the conductor node. This error might occur because a classpath required for the XML stage might not be set in case JVM options specified at the Java Integration stage are used to create the JVM.

To correct both the errors: You need to verify which JVM options are required. First you set an environment variable CC_MSG_LEVEL to 2, and compile and run the job. You can see JVM options following “The JVM is started with these options =” in the job log like below.

```
CC_JNICommon.cpp:(648) The JVM is started with these options =
-Dcom.ibm.SSL.ConfigURL=file:
C:\IBM\InformationServer\ASBNode\eclipse\plugins
\com.ibm.isf.client\ssl.client.props
-Xmx256m -Djava.class.path=
C:\IBM\InformationServer\Server\DSEngine/..\DSComponents/bin/ccapi.jar;
C:\IBM\InformationServer\Server\DSEngine/..\DSComponents/bin/ccapi.jar;;
C:\IBM\InformationServer\ASBNode\lib\java\ccapi.jar;
C:\IBM\InformationServer\ASBNode\eclipse\plugins\com.ibm.isf.client\com.ibm.ws.security.crypto.jar;
C:\IBM\InformationServer\ASBNode\eclipse\plugins\com.ibm.isf.client\com.ibm.ws.ejb.thinclient_7.0.0.0.jar
-Djava.ext.dirs=C:\IBM\InformationServer\Server\DSEngine/..\ASBNode\apps\jre\lib\ext;
C:\IBM\InformationServer\Server\DSEngine/..\ASBNode\eclipse\plugins\com.ibm.isf.client
```

In the first case, you need to verify how much the JVM is required and set the environment variable CC_JVM_OVERRIDE_OPTIONS with the JVM options after changing the value of the maximum heap size with a JVM option -Xmx. In the second case, you can see JVM options required for the XML stage by running a job which has only XML stage. By comparing the JVM options you will be able to find the XML stage requires to specify a directory containing the jar files with a JVM option -Djava.ext.dirs, and override the JVM options by setting CC_JVM_OVERRIDE_OPTIONS value like below:

```
-Dcom.ibm.SSL.ConfigURL=file:
-Dcom.ibm.tools.attach.enable=no
-Xmx256m -Djava.class.path=
-C:\IBM\InformationServer\ASBNode\lib\java\ccapi.jar
-Djava.ext.dirs=C:\IBM\InformationServer\Server\DSEngine/..\DSComponents\bin;
C:\IBM\InformationServer\Server\DSEngine/..\ASBNode\apps\jre\lib\ext;
C:\IBM\InformationServer\Server\StagingArea\Installed\XMLStage\Server\Shared;
C:\IBM\InformationServer\ASBNode\eclipse\plugins\com.ibm.isf.client
```

**Java Pack related issues**

You might encounter java.lang.ClassNotFoundException when running a Java Integration stage job migrated with the Connector Migration Tool although the original Java Pack job did not throw any error.

```
Java_Client: java.lang.ClassNotFoundException:
  myClass.MyTester at
  java.net.URLClassLoader.findClass(URLClassLoader.java:434)
  at java.lang.ClassLoader.loadClass(ClassLoader.java:653)
  at java.lang.ClassLoader.loadClass(ClassLoader.java:619)
  at com.ibm.is.cc.javastage.connector.CC_JavaConnection.connect
```

Chapter 6. Troubleshooting  45
This error might occur because in Java Pack the $DSHOME directory is considered the root directory of the relative path specified at the stage property **User's Classpath** on the **Java Pack property editor** but in the Java Integration stage $DSHOME directory might not be the root directory. The subject exception occurs if the user class files were only placed at that directory relative to $DSHOME.

To correct the error, you need to specify the absolute path at the stage property **Usage - Java - Classpath** on the **Java Integration stage property editor** or copy the class files at the directory relative to the project directory $DSHOME/../Projects/<project_name>.

- **Stage capabilities related issues**
  - You might encounter the following fatal error when running a Java Integration stage job using Java Pack API:
    ```java
    JavaPackTransformer: java.lang.IllegalAccessError:
    com/ascentialsoftware/jds/Stage.setup(Ljava/lang/Object;)V
    at com.ibm.is.cc.javastage.connector.CC_JavaPackProcessor.<init> (CC_JavaPackProcessor.java: 114)
    at com.ibm.is.cc.javastage.connector.CC_JavaConnection.connect (CC_JavaConnection.java: 164)
    ```
    This error might occur because Java Integration stage runtime wrongly would invoke a Java Pack method not for Java Integration stage. You might specify a jar file named tr4j.jar for Java Pack into classpath.
    
    To correct the error, you need to remove tr4j.jar which is a jar file for Java Pack from the following:
    - CLASSPATH environment specified at the Operating System, the DataStage project properties and the job properties.
    - **Usage - Java - Classpaths** on the Java Integration stage property editor.
    - -classpath or -cp option specified at **Usage - Java - VM option** on the Java Integration stage property editor.

- **Column mapping related issues**
  - You might encounter the following fatal warning when running a Java Integration stage job:
    ```plaintext
    A wave maker cannot be emitted.
    ```
    This error might occur because in your user class, you might have tried to emit a wave marker into an output link by calling `OutputLink.writeWaveMaker()`, but, it was not allowed since it set the capability to false in `getCapabilities()`.
    
    - You might encounter the following fatal error when running a Java Integration stage job:
      ```plaintext
      Stage design is incompatible with Java code.
      Stage has 0 output links but Java code expects min.=1 max.=1.
      ```
      This error might occur because in your user class, you expected the Java Integration stage must have 1 output link, but the Java Integration stage did not have any output link on your job design.
      
      To fix the first issue, you need to modify your Java code to solve a conflict of the capability. To fix the second issue, you need to modify your job design based on the capabilities defined in your Java code.
– You might encounter the following fatal error when running a Java Integration stage job using JavaBeans:

    com.ascential.e2.common.CC_Exception: Java bean could not be set.
    Class name: userClass
    The mapping from columns to bean properties was not defined.
    Launch the Column mapping editor to define the column mappings for the bean.
    Bean class name: userInputBean
    at com.ibm.is.cc.javastage.connector.CC_JavaLinkImpl.setBeanInfo
    (CC_JavaLinkImpl.java: 146)
    at com.ibm.is.cc.javastage.connector.CC_JavaInputLinkImpl.setBeanClass
    (CC_JavaInputLinkImpl.java: 128)
    at com.ibm.is.cc.javastage.connector.CC_JavaAdapter.updateLinkWithBeanInfo
    (CC_JavaAdapter.java: 208)
    at com.ibm.is.cc.javastage.connector.CC_JavaAdapter.preRunNode
    (CC_JavaAdapter.java: 421)

This error might occur because you specified userInputBean for an input link by coding `getBeanForInput()` in your user class userClass, but you did not set column mapping for that link.

– You might encounter the following warning when running a Java Integration stage job using JavaBeans:

    com.ascential.e2.common.CC_Exception: Java bean could not be set.
    Class name: userOutputBean
    Type mismatch occurred. Column name: aaa, Column type class: java.sql.Date,
    Bean property name: bbb, Bean property type class: java.lang.String,
    Bean class name: userOutputBean
    at com.ibm.is.cc.javastage.connector.CC_JavaLinkImpl.setBeanInfo
    (CC_JavaLinkImpl.java: 146)
    at com.ibm.is.cc.javastage.connector.CC_JavaOutputLinkImpl.setBeanClass
    (CC_JavaOutputLinkImpl.java: 278)
    at com.ibm.is.cc.javastage.connector.CC_JavaAdapter.updateLinkWithBeanInfo
    (CC_JavaAdapter.java: 218)
    at com.ibm.is.cc.javastage.connector.CC_JavaAdapter.preRunNode
    (CC_JavaAdapter.java: 421)

This error might occur because you might have specified userOutputBean for an output link and also configured that link, but, you might have selected a wrong DataStage column aaa of which column type is java.sql.Date and map it into the Java bean property bbb of which type is java.lang.String.

To correct the error: You need to verify the link property **Usage - Column mapping** and update it by clicking the **Configure** button. You can see the current setting on the **Column Mapping Editor** and modify the mapping by selecting any other candidate from the pull down list of each DataStage column. Also, if there is none of the column mapping or you want to see all the JavaBean properties then click the **Browse Objects** button. You can see and select them by clicking check boxes on the **Select Bean Properties** dialog. If you want to save new selections, click the **OK** button on the **Select Bean Properties** dialog and click the **Finish** button on the **Column Mapping Editor**.

**Link related issues**

– You might encounter the following fatal error when running a Java Integration stage job with multiple input links:

    [Input link 0] com.ascential.e2.common.CC_Exception: User code returned the bean class "YourInput" for this link, but this is not matched to "MyInput" specified in the design.
    Launch the Column mapping editor to reconfigure the column mappings.
    at com.ibm.is.cc.javastage.connector.CC_JavaInputLinkImpl.setBeanClass
    (CC_JavaInputLinkImpl.java: 123)
    at com.ibm.is.cc.javastage.connector.CC_JavaAdapter.updateLinkWithBeanInfo
    (CC_JavaAdapter.java: 208)
    at com.ibm.is.cc.javastage.connector.CC_JavaAdapter.preRunNode
    (CC_JavaAdapter.java: 421)
This error might occur because you might have changed the link order after configuring the column mapping.

- You might encounter the following fatal error when running a Java Integration stage job with reject links:

  ```java
  com.ascential.e2.common.CC_Exception: Reject link is not configured.
  Configure reject link and save it.
  at com.ibm.is.cc.javastage.connector.CC_JavaRecordDataSetConsumer.setRejectManager
  (CC_JavaRecordDataSetConsumer.java: 182)
  at com.ibm.is.cc.javastage.connector.CC_JavaAdapter.getRejectDataSetProducer
  (CC_JavaAdapter.java: 349)
  ```

To correct the errors:

For the error mentioned first, you need to click the **Configure** button on the **Java Integration stage property editor** and configure the column mapping again. You always need to verify the link order on the **Link Ordering** tab on the **Java Integration stage property editor** if your job has multiple input and output links to prevent any inconsistencies between your job design and Java code.

For the error mentioned second, you need to open the **Reject** tab and confirm the right input row was selected at the stage property **Reject - From Link** on the **Java Integration stage property editor**.
Chapter 7. Environment variables: Java Integration stage

The Java Integration stage uses these environment variables.

**CC_IGNORE_TIME_LENGTH_AND_SCALE**

Set this environment variable to change the behavior of the connector on the parallel canvas.

When this environment variable is set to 1, the connector running with the parallel engine ignores the specified length and scale for the timestamp column. For example, when the value of this environment variable is not set and if the length of the timestamp column is 26 and the scale is 6, the connector on the parallel canvas considers that the timestamp has a microsecond resolution. When the value of this environment variable is set to 1, the connector on the parallel canvas does not consider that the timestamp has a microsecond resolution unless the microseconds extended property is set even if the length of the timestamp column is 26 and the scale is 6.

**CC_JNI_EXT_DIRS**

Set this environment variable to add a prefix to the class path of java.ext.dirs system property.

When the value of this environment variable is set, a prefix is added to the class path of java.ext.dirs system property.

**CC_JVM_OPTIONS**

Set this environment variable to specify the JVM arguments that are used when a job is run.

When this variable is specified, it takes precedence over the value of the default JVM arguments for the Java-based connectors. For example, if you set **CC_JVM_OPTIONS** as `-Xmx512M`, the maximum heap size is set to 512 MB when JVM instances for the connector are created.

**CC_JVM_OVERRIDE_OPTIONS**

Set this environment variable to override the JVM options for the conductor node so that you can avoid or fix a possible conflict.

In the conductor node in a parallel job, Java connectors are initialized for schema reconciliation. Therefore, all Java connectors in a job are initialized in the same JVM. In a single job, multiple stages might be developed in Java. Each of these stages might define JVM options such as class path, system property, heap size and so on. If two stages are run in the same physical JVM, the JVM options might conflict with each other.

**CC_MSG_LEVEL**

Set this environment variable to specify the minimum severity of the messages that the connector reports in the log file.
At the default value of 3, informational messages and messages of a higher severity are reported to the log file.

The following list contains the valid values:

- 1 - Trace
- 2 - Debug
- 3 - Informational
- 4 - Warning
- 5 - Error
- 6 - Fatal

**JAVASTAGE_API_DEBUG**

Set this environment variable to control whether the debug messages that are specified by the `com.ibm.is.cc.javastage.api.Logger.debug()` API are reported to the Director log file.

When the value of this variable is 1, debug messages are reported to the log file. If the value of this environment variable is not 1, all of the debug messages are ignored. For more details, see the Javadoc information for the Java Integration stage API.
Appendix A. Product accessibility

You can get information about the accessibility status of IBM products.

The IBM InfoSphere Information Server product modules and user interfaces are not fully accessible. The installation program installs the following product modules and components:

- IBM InfoSphere Business Glossary
- IBM InfoSphere Business Glossary Anywhere
- IBM InfoSphere DataStage
- IBM InfoSphere FastTrack
- IBM InfoSphere Information Analyzer
- IBM InfoSphere Information Services Director
- IBM InfoSphere Metadata Workbench
- IBM InfoSphere QualityStage

For information about the accessibility status of IBM products, see the IBM product accessibility information at http://www.ibm.com/able/product_accessibility/index.html.

Accessible documentation

Accessible documentation for InfoSphere Information Server products is provided in an information center. The information center presents the documentation in XHTML 1.0 format, which is viewable in most Web browsers. XHTML allows you to set display preferences in your browser. It also allows you to use screen readers and other assistive technologies to access the documentation.

The documentation that is in the information center is also provided in PDF files, which are not fully accessible.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility.
Appendix B. Reading command-line syntax

This documentation uses special characters to define the command-line syntax.

The following special characters define the command-line syntax:

- `[ ]` Identifies an optional argument. Arguments that are not enclosed in brackets are required.
- `...` Indicates that you can specify multiple values for the previous argument.
- `|` Indicates mutually exclusive information. You can use the argument to the left of the separator or the argument to the right of the separator. You cannot use both arguments in a single use of the command.
- `{ }` Delimits a set of mutually exclusive arguments when one of the arguments is required. If the arguments are optional, they are enclosed in brackets ([]).

Note:
- The maximum number of characters in an argument is 256.
- Enclose argument values that have embedded spaces with either single or double quotation marks.

For example:

```
wssetsrc [-S server] [-l label] [-n name] source
```

The `source` argument is the only required argument for the `wssetsrc` command. The brackets around the other arguments indicate that these arguments are optional.

```
wlsac [-l | -f format] [key...] profile
```

In this example, the `-l` and `-f format` arguments are mutually exclusive and optional. The `profile` argument is required. The `key` argument is optional. The ellipsis (...) that follows the `key` argument indicates that you can specify multiple key names.

```
wrb -import {rule_pack | rule_set}...
```

In this example, the `rule_pack` and `rule_set` arguments are mutually exclusive, but one of the arguments must be specified. Also, the ellipsis marks (...) indicate that you can specify multiple rule packs or rule sets.
Appendix C. How to read syntax diagrams

The following rules apply to the syntax diagrams that are used in this information:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
  - The >>--- symbol indicates the beginning of a syntax diagram.
  - The ---> symbol indicates that the syntax diagram is continued on the next line.
  - The >--- symbol indicates that a syntax diagram is continued from the previous line.
  - The -->< symbol indicates the end of a syntax diagram.
- Required items appear on the horizontal line (the main path).

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>required_item</td>
</tr>
</tbody>
</table>
```

- Optional items appear below the main path.

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>required_item</td>
</tr>
</tbody>
</table>
```

If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>required_item</td>
</tr>
</tbody>
</table>
```

- If you can choose from two or more items, they appear vertically, in a stack. If you must choose one of the items, one item of the stack appears on the main path.

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>required_item</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

If choosing one of the items is optional, the entire stack appears below the main path.

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>required_item</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

If one of the items is the default, it appears above the main path, and the remaining choices are shown below.

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>required_item</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

- An arrow returning to the left, above the main line, indicates an item that can be repeated.
If the repeat arrow contains a comma, you must separate repeated items with a comma.

A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.

**Fragment-name:**

- Keywords, and their minimum abbreviations if applicable, appear in uppercase. They must be spelled exactly as shown.
- Variables appear in all lowercase italic letters (for example, `column-name`). They represent user-supplied names or values.
- Separate keywords and parameters by at least one space if no intervening punctuation is shown in the diagram.
- Enter punctuation marks, parentheses, arithmetic operators, and other symbols, exactly as shown in the diagram.
- Footnotes are shown by a number in parentheses, for example (1).
Appendix D. Contacting IBM

You can contact IBM for customer support, software services, product information, and general information. You also can provide feedback to IBM about products and documentation.

The following table lists resources for customer support, software services, training, and product and solutions information.

Table 3. IBM resources

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description and location</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM Support Portal</td>
<td>You can customize support information by choosing the products and the topics that interest you at <a href="http://www.ibm.com/support/entry/portal/Software/Information_Management/InfoSphere_Information_Server">www.ibm.com/support/entry/portal/Software/Information_Management/InfoSphere_Information_Server</a></td>
</tr>
<tr>
<td>Software services</td>
<td>You can find information about software, IT, and business consulting services, on the solutions site at <a href="http://www.ibm.com/businesssolutions/">www.ibm.com/businesssolutions/</a></td>
</tr>
<tr>
<td>My IBM</td>
<td>You can manage links to IBM Web sites and information that meet your specific technical support needs by creating an account on the My IBM site at <a href="http://www.ibm.com/account/">www.ibm.com/account/</a></td>
</tr>
<tr>
<td>Training and certification</td>
<td>You can learn about technical training and education services designed for individuals, companies, and public organizations to acquire, maintain, and optimize their IT skills at <a href="http://www.ibm.com/software/sw-training/">http://www.ibm.com/software/sw-training/</a></td>
</tr>
</tbody>
</table>
Appendix E. Accessing and providing feedback on the product documentation

Documentation is provided in a variety of locations and formats, including in help that is opened directly from the product client interfaces, in a suite-wide information center, and in PDF file books.

The information center is installed as a common service with IBM InfoSphere Information Server. The information center contains help for most of the product interfaces, as well as complete documentation for all the product modules in the suite. You can open the information center from the installed product or from a Web browser.

**Accessing the information center**

You can use the following methods to open the installed information center.

- Click the **Help** link in the upper right of the client interface.

  **Note**: From IBM InfoSphere FastTrack and IBM InfoSphere Information Server Manager, the main Help item opens a local help system. Choose **Help > Open Info Center** to open the full suite information center.

- Press the F1 key. The F1 key typically opens the topic that describes the current context of the client interface.

  **Note**: The F1 key does not work in Web clients.

- Use a Web browser to access the installed information center even when you are not logged in to the product. Enter the following address in a Web browser: http://host_name:port_number/infocenter/topic/com.ibm.swg.im.iis.productization.iisinfsv.home.doc/ic-homepage.html. The host_name is the name of the services tier computer where the information center is installed, and port_number is the port number for InfoSphere Information Server. The default port number is 9080. For example, on a Microsoft® Windows® Server computer named iisdocs2, the Web address is in the following format: http://iisdocs2:9080/infocenter/topic/com.ibm.swg.im.iis.productization.iisinfsv.nav.doc/dochome/iisinfsrv_home.html.


**Obtaining PDF and hardcopy documentation**

- A subset of the PDF file books are available through the InfoSphere Information Server software installer and the distribution media. The other PDF file books are available online and can be accessed from this support document: [https://www.ibm.com/support/docview.wss?uid=swg27008803&wv=1](https://www.ibm.com/support/docview.wss?uid=swg27008803&wv=1)

- You can also order IBM publications in hardcopy format online or through your local IBM representative. To order publications online, go to the IBM Publications Center at [http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss](http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss)
Providing comments on the documentation

Your feedback helps IBM to provide quality information. You can use any of the following methods to provide comments:

- To comment on the information center, click the Feedback link on the top right side of any topic in the information center.
- Send your comments by using the online readers’ comment form at [www.ibm.com/software/awdtools/rcf/](http://www.ibm.com/software/awdtools/rcf/).
- Send your comments by e-mail to comments@us.ibm.com. Include the name of the product, the version number of the product, and the name and part number of the information (if applicable). If you are commenting on specific text, include the location of the text (for example, a title, a table number, or a page number).
Notices and trademarks

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

Notices

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 character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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 or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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

COPYRIGHT LICENSE:

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

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

**Trademarks**

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

The following terms are trademarks or registered trademarks of other companies:

Adobe is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

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

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

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

The United States Postal Service owns the following trademarks: CASS, CASS Certified, DPV, LACSlink, ZIP, ZIP + 4, ZIP Code, Post Office, Postal Service, USPS and United States Postal Service. IBM Corporation is a non-exclusive DPV and LACSlink licensee of the United States Postal Service.

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

A
API 18

C
column definitions
setting up 31, 34, 37
command-line syntax
conventions 53
commands
syntax 53
Connector Migration Tool 39
customer support
contacting 57

D
data
looking up 38
retrieving 30
data types
DataStage 12, 13
importing data 13
Java Integration stage 12, 13
loading data 12
reading data 13
writing data 12

E
environment variables
Java Integration stage 49

J
Java code 11
capabilities 7
compiling code 6
Configuration class 6
Executing on parallel engine 6
Implementing abstract methods of Processor class 3
passing data 32
writing records 9
Java Integration stage 14, 18
accessing 29
compiling and running jobs 41
configuring as a source 30
configuring as a target 33
configuring as a transformer 36
data
transforming 35
InputLink interface 8
Java 6 SDK 3
Java code 3
java.sql.Time 13, 17
microsecond resolution 13, 17
overview 1
reading records 8

M
mapping
data types 12, 13
link index 32, 35, 38
migrating to use Java Integration stage 39

P
product accessibility
accessibility 51
product documentation
accessing 59

R
reference links 38
rejecting records 11
retrieving data 30

S
software services
contacting 57
special characters
in command-line syntax 53
stage operations
setting up column definitions 31, 34, 37
support
customer 57
syntax
command-line 53

T
trademarks
list of 61

Troubleshooting
Java Integration stage 43

U
using user-defined properties 14

V
validation
running 41

W
web sites
non-IBM 55