IBM InfoSphere DataStage and QualityStage
Version 11 Release 3

# *Guide to Integrating Java Code*

SC19-4336-00

IBM InfoSphere DataStage and QualityStage
Version 11 Release 3

# *Guide to Integrating Java Code*

SC19-4336-00

**Note**

Before using this information and the product that it supports, read the information in "Notices and trademarks" on page 75.

# Contents

# 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:
   - `<ISDIR>/Server/DSComponents/bin` , if you are using an Information Server engine tier machine.
   - `<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

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()`
- `validateConfiguration()`

- getConfigurationErrors()
- getBeanForInput()
- getBeanForOutput()
- getAdditionalOutputColumns()
- initialize()
- process()
- terminate()
- getColumnMetadataForInput()
- getColumnMetadataForOutput()
- getUserPropertyDefinitions()

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.

```
package samples;

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

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());
      }
```

```
    }
    while (true);
  }
}
```

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.

```
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.

```
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.

```
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.

```
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

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**

- `getDataChannel()`
- `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()`

**Methods for accessing data transfer services**

- `getDataChannel()`

## 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()`
- `getColumnTransferBehavior()`
- `isWaveGenerator()`
- `isRunOnConductor()`
- `setMinumumInputLinkCount()`
- `setMaximumInputLinkCount()`
- `setMinimumOutputStreamLinkCount()`
- `setMaximumOutputStreamLinkCount()`
- `setMinimumRejectLinkCount()`
- `setMaximumRejectLinkCount()`
- `setIsWaveGenerator()`
- `setColumnTransferBehavior()`
- `setIsRunOnConductor()`

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.

```
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.

```
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.

```
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**

- getObject()
- 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.

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

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

```
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.

```
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
    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);
```

.

## Looking up data in the Sparse lookup mode

The Java Integration stage supports sparse lookup. You can implement the lookup logic in your Java code to read the input record and write to the output record one by one.

When the connector is used to complete a sparse lookup operation, the connector has one input link, one output link, and an optional reject link. The connector with a single output reference link is internally converted to contain one input link, one output link and optionally one reject link, if the Lookup stage in the job contains a reject link. You must add an implementation in your Java code to run in sparse lookup mode in the `process()` method.

To identify the lookup mode in the `process()` mode, call the `getSparseLookupMode()` method for the Configuration interface as follows:

```
SparseLookupMode m_sparseLookupMode = m_configuration.getSparseLookupMode()
```

The return value indicates the lookup mode that is configured. In the Lookup Stage Conditions pane, the Lookup Failure column indicates the expected behavior when the lookup operation fails. The Java code must contain the logic to create a loop to repeatedly read the data from the input link, and write to the output link or reject link. The schema for the input link and reject link is the same as the schema for the input link for the Lookup stage. The schema for the output link is the same as the schema for the output reference link for the Java Integration stage. The Java code must read the key columns on the input link and write the output records that have the lookup data.

To identify the key columns on the input link, use the `isKey()` method for the ColumnMetadata interface as follows:

```
List<ColumnMetadata> inputColumns = m_inputLink.getColumnMetadata();
ColumnMetadata cm = inputColumns.get(columnIndex);
boolean isKey = cm.isKey();
```

After the Java code writes output records, the Java Integration stage and the framework add or copy the additional columns according to the mapping configuration in the Lookup stage so that the schema of the output link is the same as the schema of the output link from the Lookup stage.

When you implement the logic for sparse lookup, ensure that the user code calls the `writeRecord()` method for OutputLink or RejectLink, before it calls the `InputLink.readRecord()` method to fetch the next incoming record. The framework also writes to the column for sparse lookup output along with the connector. The connector framework holds the cursor for the input record that the user code reads and when you call the `InputLink.readRecord()` method, it changes the cursor position to a different record. To write the output record correctly, the `writeRecord()` method must be called before the `InputLink.readRecord()` method for the next record.

It is not mandatory to implement an expected behavior for sparse lookup on lookup failure. However, it is a good practice to implement an expected behaviour as the user code can check the return value for the `getSparseLookupMode()` methods and process failed records. The following behavior is expected for each mode:

*Table 1. Expected behaviour for sparse lookup on lookup failure*

| Mode | Expected behavior |
|---|---|
| Continue | The lookup operation continues after the empty data is sent to columns in the looked up data. You can implement Java code to pass empty records or record with no values to the `writeRecord()` method. |
| Drop | Drop the records that don't match the records on lookup table. When the code makes the next call for the `readRecord()` method, the input record is discarded automatically. There is nothing to be implemented in the Java code. |
| Fail | The job ends when the lookup fails. The Java code throws an exception when the job ends. |
| Reject | The records that don't match the records on the lookup table are rejected. The input record are passed to the `RejectLink.writeRecord()` method. |

The following sample code shows the `process()` method and the implementation of expected behavior for sparse lookup on lookup failure:

```
public void process() throws Exception
{

    if (m_sparseLookupMode == Configuration.SparseLookupMode.NOT_SPARSE)
    {
         // regular implementation for process()
         // ...
         // ...
    }
    else
    {
         // Sparse lookup implementation
         do
         {
            InputRecord inputRecord = m_inputLink.readRecord();
            if (inputRecord == null)
            {
               // No more input
               return;
```

```
            }

            OutputRecord outputRecord = m_outputLink.getOutputRecord();
            OutputRecord emptyRecord = m_outputLink.getOutputRecord();

            // -- do the lookup and fill output record
            boolean fLookupFail = lookup(inputRecord, outputRecord);

            if (!fLookupFail)
            {
               // lookup success - put the output record populated as you like
               m_outputLink.writeRecord(outputRecord);
            }
            else
            {
               switch(m_sparseLookupMode)
               {
               case Configuration.SparseLookupMode.CONTINUE:
                  // "Continue" mode - send out the empty record
                  m_outputLink.writeRecord(emptyRecord);
                  break;
               case Configuration.SparseLookupMode.DROP:
                  // "Drop" mode - do nothing and proceed to read next record.
                  break;
               case Configuration.SparseLookupMode.FAIL:
                  // "Fail" mode - throw exception to abort the job
                  throw new ConnectorException(9998, "Lookup failed.");
               case Configuration.SparseLookupMode.REJECT:
                  // "Reject" mode - copy input record into reject link
                  RejectRecord rejectRecord = m_rejectLink.getRejectRecord
(inputRecord);
                  m_rejectLink.writeRecord(rejectRecord);
                  break;
               }
            }
         }
         while(true);
      }
}
```

In the sample code, users implement their own lookup logic in the `lookup()` function by looking at the input data on the `inputRecord` argument and populate the `outputReocrd` argument with the extracted data. The return value indicates whether the lookup is a success or a failure.

The sparse lookup mode runs with the special link topology. In the sparse lookup mode, the Java Integration Stage does not enforce the capability check on the number of input and reject links.

## 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 2. InfoSphere DataStage data types and their equivalent Java data types*

| InfoSphere DataStage data types | Java data types |
|---|---|
| BigInt | java.math.BigInteger |
| Binary | byte[] |
| Bit | int/java.lang.Integer or boolean/ java.lang.Boolean<br>**Note:** boolean/java.lang.Boolean is only applicable for Java bean or UDF case |
| Char | java.lang.String |
| Date | java.sql.Date |
| Decimal | java.math.BigDecimal |
| Double | double/java.lang.Double |
| Float | float/java.lang.Float |
| Integer | long/java.lang.Long |
| LongNVarChar | java.lang.String |
| LongVarBinary | byte[] |
| LongVarChar | java.lang.String |
| NChar | java.lang.String |
| Numeric | java.math.BigDecimal |
| NVarChar | java.lang.String |
| Real | java.lang.Float |
| SmallInt | java.lang.Integer |
| Time | java.sql.Time |
| Timestamp | java.sql.Timestamp |
| TinyInt | java.lang.Short |
| VarBinary | byte[] |
| VarChar | java.lang.String |

## 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 3. Java data types and their equivalent InfoSphere DataStage data types*

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

## 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 wants 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()
- getNativeType()
- getPrecision()
- getScale()
- getSQLType()
- getSQLTypeName()
- getType()
- hasMicrosecondResolution()
- isKey()
- isNullable()
- isSigned()
- isUnicode()

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

```
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 meta data 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.

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

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.

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

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.

| **Java type** | **SQL type** |
|---|---|
| byte[] | Binary |
| boolean/java.lang.Boolean | Bit |
| short/java.lang.Short | TinyInt |
| int/java.lang.Integer | SmallInt |
| double/java.lang.Double | Double |
| float/java.lang.Float | Float |
| long/java.lang.Long | Integer |
| java.lang.String | VarChar |
| java.math.BigInteger | BigInt |
| java.math.BigDecimal | Decimal |
| java.sql.Date | Date |
| java.sql.Time | Time |

| Java type | SQL type |
|---|---|
| com.ibm.is.cc.javastage.api.TimeMicroseconds.class | Time |
| java.sql.Timestamp | Timestamp |

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.

## Running the Java code on the conductor node

The Java Integration stage supports the execution of Java code on the conductor node during initialization and termination of stages.

To execute the Java code on the conductor, your Java code calls the `setIsRunOnConductor` method of the `Capabilities` class. During the initialization of the stage, the `Processor` class calls the following methods:

1. `getCapabilities()`
2. `validateConfiguration()`
3. `initialize()`

During initialization, some of the additional methods of the `Processor` class are called in the following scenarios:

- If the `validateConfiguration` method returns false, the `getConfigurationErrors` method is called.
- If runtime column propagation is enabled on one output link and the `getBeanForOutput` method is implemented to return the JavaBean object, then the following methods of the `Processor` class are called during initialization:
  1. `getCapabilities()`
  2. `validateConfiguration()`
  3. `getBeanForOutput()`
  4. `getAdditionalOutputColumns()`
- If the capability that runs of your Java code on the conductor node and runtime column propagation are both enabled on at least one output link associated with the JavaBean object, then the following methods of the `Processor` class are called in the specified order during initialization. The `validateConfiguration` method is called twice on the conductor node.
  1. `getCapabilities()`
  2. `validateConfiguration()`

3. getBeanForOutput()
4. getAdditionalOutputColumns()
5. validateConfiguration()
6. initialize()

During the termination of a stage, the terminate() method of the Processor class is called.

The following example Java code is used to set the capability to run the methods of the Processor class on the conductor node for initialization and termination of stage.

```
public Capabilities getCapabilities()
{
   Capabilities capabilities = new Capabilities();
   capabilities.setIsRunOnConductor(true);
   return capabilities;
}
```

The Java code checks the return value of the getNodeNumber method of the Configuration class to identify whether the method is called on the conductor node or player nodes. If the return value is -1 the method runs on the conductor node. If the return value is any other number, for example 0 or 1, the method is invoked on player nodes.

In the following example, the Processor class completes initialization and termination processes on the conductor node and player nodes.

```
private int m_nodeID = -1;
public boolean validateConfiguration(Configuration configuration,
boolean isRuntime) throws Exception
{
...
m_nodeID = configuration.getNodeNumber();
...
return true;
}

public void initialize() throws Exception
{
if (m_nodeID == -1) // On Conductor
   {
     // Perform any initialization process on the conductor node
   }
   else
   {
     // Perform any initialization process on each player node
   }
}

public void terminate(boolean isAborted) throws Exception
{
   if (m_nodeID == -1)  // On Conductor
   {
    // Perform any termination process on the connector node
   }
   else
   {
   // Perform any termination process on each player node
   }
}
```

## Transferring data from the conductor node to player nodes

The Java Integration stage provides your Java code with the DataChannel interface to transfer data from the conductor node to the player nodes.

The DataChannel interface provides the following methods:

- sendTo()
- receiveFrom()

To get an instance of the DataChannel interface, your Java code calls the getDataChannel method of the Configuration class. To send data from the conductor node, your Java code calls the sendTo method. To receive data on each player node, your Java code calls the receiveFrom method.

To enable the data transfer service capability during the stage initialization and termination processes, your Java code calls the setIsRunOnConductor method of the Capabilities class. The getDataChannel method returns null if the capability is not enabled.

In the following example code, Java code passes string data from the conductor node to player nodes.

```
private int m_nodeID = -1;

public boolean validateConfiguration(Configuration configuration,
boolean isRuntime) throws Exception
{
   ...
   m_nodeID = configuration.getNodeNumber();
   if (isRuntime == true)
   {
      // Get data channel
      DataChannel channel = configuration.getDataChannel();
      if (m_nodeID == -1) // On Conductor node
      {
         // Send data to the player node
         String object = new String("*** test data ***");
         Logger.information(object + " is sent from Conductor to Player. ");
         channel.sendTo(DataChannel.NODE_PLAYER, object);
      }
      else  // On Player node
      {
         // Receive data from the conductor node
         String object = (String) channel.receiveFrom(DataChannel.NODE_CONDUCTOR);
         Logger.information(object + "is received on Player(" +  m_nodeID + ").");
      }
   }
   ...
   return true;
}
```

When the example code is run, the following messages are logged in the InfoSphere DataStage job log:

```
Java_Stage: *** test data *** is set on the Conductor node.
Java_Stage,0: *** test data *** is received on Player(0).
Java_Stage,1: *** test data *** is received on Player(1).
```

## Logging messages with the Java Integration stage

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

At run time, error, warning, and informational messages are written to the job log. For example, your code can write 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, which you can view in the InfoSphere Information Server Web console. For example, your 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.

The Logger class provides the following methods:

- `debug(int messageNumber, String message)`
- `debug(String message)`
- `fatal(String message)`
- `getComponentID()`
- `information(int messageNumber, String message)`
- `information(String message)`
- `isDebugEnabled ()`
- `setComponentID(String compID)`
- `warning(int messageNumber, String message)`
- `warning(String message)`

For more information about the Logger class, see the Javadoc for the Java Integration stage API.

## Message ID format for the Logger class

The Logger class (com.ibm.is.cc.javastage.api.Logger) logs messages in the `IIS-`*`categoryID-componentID-messageNumber`* format.

IIS is the prefix for InfoSphere Information Server. *categoryID* is the category of the component that produces the message. CONN is the *categoryID* for connectors, including the Java Integration stage. *componentID* is the component throwing the message. *messageNumber* is a positive integer for the message.

The following table contains a list of event types and the predefined message IDs in the Java Integration stage.

*Table 4. List of event types and predefined message IDs in the Java Integration stage*

| Method | Event type | Message ID |
|---|---|---|
| Logger.debug(String message) | Informational | IIS-CONN-JAVA-00011 |
| Logger.information(String message) | Informational | IIS-CONN-JAVA-00012 |
| Logger.warning(String message) | Warning | IIS-CONN-JAVA-00013 |
| Logger.fatal(String message) | Fatal | IIS-CONN-JAVA-00015 |

## Logging messages with custom IDs

If the Logger class is invoked at run time, the messages are logged to the Director client log. If the Logger class is invoked at design time, the messages are logged to the Connector Access Service log. At run time, you can log a message with the

message ID that is defined in the Java Integration stage or log a message with a custom message ID that is defined in your Java code.

You can use the following methods of the Logger class in your Java code to log messages with custom message IDs to the Director client log:

- `setComponentID(String compID)`
- `debug(int messageNumber, String message)`
- `information(int messageNumber, String message)`
- `warning(int messageNumber, String message)`

For more information about the Logger class, see the Javadoc information for the Java Integration stage API.

In the following example an informational message `Job completed successfully` is logged with the message ID IIS-CONN-MYCONN-00123.

```
Logger.setComponentID("MYCONN");
Logger.information(123, "Job completed successfully");
```

## Logging debug messages

You can use Java code to log debug messages when the JAVASTAGE_API_DEBUG environment variable is set to 1.

You can use the following methods of the `Logger` class in your Java code to log debug messages:

- `debug(String message)`
- `debug(int messageNumber, String message)`
- `isDebugEnabled()`
- `setComponentID(String compID)`

The following example logs the debug messages:

```
debug test 001
debug test 002
 ...
debug test 999
```

```
if (Logger.isDebugEnabled())
{
   Logger.debug("debug test 001");
   Logger.debug("debug test 002");
   ...
   Logger.debug("debug test 999");
}
```

The `Logger.isDebugEnabled()` method returns true if the JAVASTAGE_API_DEBUG environment variable is set to 1. To avoid calling the `Logger.debug` method unnecessarily, ensure that the `Logger.isDebugEnabled()` method returns true.

# Terminating a job from the Java code

You can throw an Exception class object, or throw a ConnectorException class object, or call the Logger.fatal() method to stop a job before it ends naturally.

## Throwing an Exception class object (Java Integration stage)

You can use the `Exception (java.lang.Exception)` class or its subclass object from your Processor implementation to stop a job.

The following example shows Java code that does a basic calculation to get a price. If the value set to `count` is zero, the Java code throws the `java.lang.ArithmeticException` exception and logs the message: `Divide by zero`.

```
public void process() throws Exception
{
   int count = 0;
   int total = 1000;
   ...
   int price = total / count;
}
```

The Java code can also construct an `ArithmeticException` object with a message that you specify and throw it to terminate a job.

```
public void process() throws Exception
{
   ...
   if (result < 0)
   {
      throw new ArithmeticException("An exception occurred. result=" + result);
   }
}
```

## Throwing a ConnectorException class object (Java Integration stage)

You can use the ConnectorException(com.ibm.is.cc.javastage.api.ConnectorException) class object from Processor implementation to stop a job that is running or to end a design-time operation with a message.

Use the `ConnectorException` class to log messages that do not include the stack trace information and also to assign message IDs to those messages.

In the following example, Java code constructs and throws a `ConnectorException` object after calling the Logger.setComponentID() method. The connector framework catches the exception, logs the error message `An exception occurred. result=-1` and stops the job. The message ID is IIS-CONN-MYCONN-00999.

```
public class MyConnector extends Processor
{
   public MyConnector()
   {
     super();
     Logger.setComponentID("MYCONN");
   }


   public void process() throws Exception
   {
      ...
      if (result < 0)
      {
         throw new ConnectorException(999, " An exception
occurred. result=" + result);
      }
   }
   ...
}
```

## Calling the Logger.fatal() method

You can terminate a job by calling the com.ibm.is.cc.javastage.api.Logger.fatal() method from your Processor implementation.

When Java code calls the Logger.fatal() method, the job does not return the control to the Java code. The `Processor.terminate()` method, which stops the job cannot be invoked. This might result in an issue while restarting the job.

If your Java code requires a termination process, then use the `ConnectorException` object or any other Exception object, because the Processor.terminate() method is invoked with the `isAborted` argument set to true.

The following example shows Java code calls the Logger.fatal() method. The job ends and the message`My job terminates abnormally` is logged to the job log with the message ID IIS-CONN-JAVA-00015.

```
Logger.fatal("My job terminates abnormally.");
```

# 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**

```
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 proceessing.
    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".
        // 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

```
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;
}

/**
 * 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 theOutputRecord 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:

```
public class UserDefinedFunction
{

   /**
```

```
    * Passes primitive type double and a bean as UDF arguments and
* returns a bean.
    *
    * @param commission commission
    * @param input {@link InputBean} object.
    * @return output {@link 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. Java Integration stage API

Use the Java Integration stage API to write your Java code.

For more information, see the Java Integration stage API topic in IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSZJPZ_11.3.0/com.ibm.swg.im.iis.ds.javastage.api.doc/javadoc/index.html).

# Chapter 4. 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**.

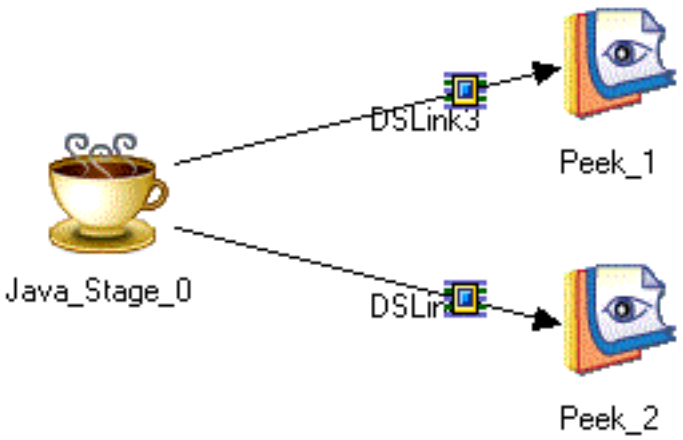



*Figure 1. Example of reading data*

### 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:

| Option | Description |
|---|---|
| **Method 1** | 1. In the **Column name** column, double-click inside the appropriate cell and type a column name.<br>2. For each cell in the row, double-click inside the cell and select the options that you want.<br>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.<br>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 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.

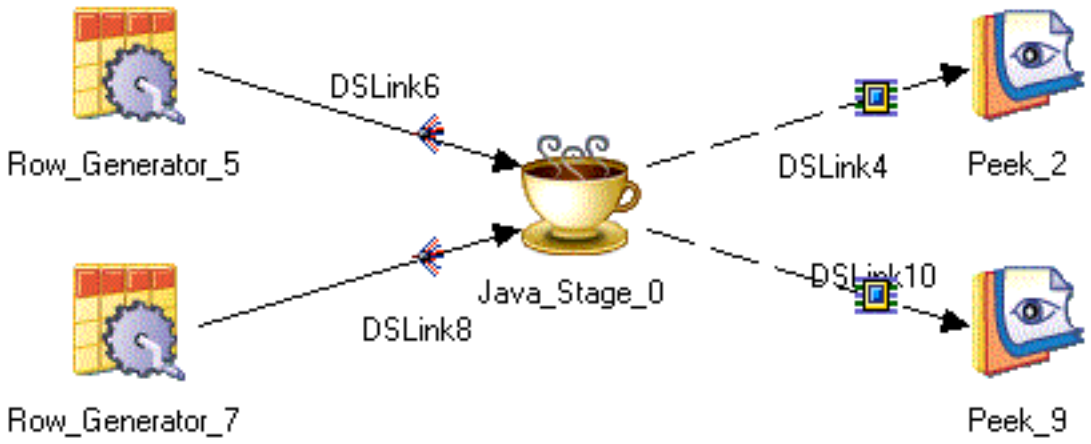



*Figure 2. Example of writing data*

## 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:

| Option | Description |
|---|---|
| **Method 1** | 1. In the **Column name** column, double-click inside the appropriate cell and type a column name.<br>2. For each cell in the row, double-click inside the cell and select the options that you want.<br>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.<br>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**.

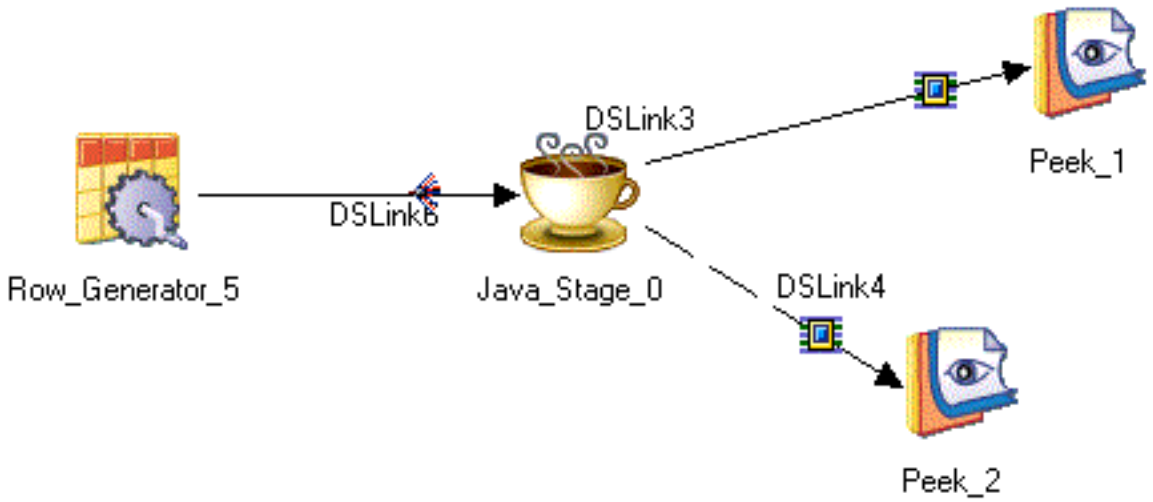



*Figure 3. Example of transforming data*

## 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.
   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 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:

| Option | Description |
|---|---|
| **Method 1** | 1. In the **Column name** column, double-click inside the appropriate cell and type a column name.<br>2. For each cell in the row, double-click inside the cell and select the options that you want.<br>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.<br>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

You can look up data by using a reference link to link the Java Integration stage to a Lookup stage.

### About this task

You can select the lookup type as Normal or Sparse in the stage editor. For a normal lookup, your Java code does not need any implementation as the output reference link is treated same as output stream link. However, sparse lookup internally changes the link topology in such a way that the connector contains one input link, one output link, and an optional reject link.When Sparse lookup is selected, make sure that only one output reference link is connected to the Java Integration stage.

### 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**.
5. Double click the Java Integration stage icon.
6. In the **Properties** tab on the **Stage** tab, select either **Normal** or **Sparse**.
7. Click **OK**.

# Chapter 5. 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 6. 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 7. API samples

Use the API samples to create sample jobs for the Java Integration stage.

For more information, see the API Sample topic in IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSZJPZ_11.3.0/com.ibm.swg.im.iis.ds.javastage.api.doc/SampleGettingStarted.html).

# Chapter 8. Troubleshooting the Java Integration stage

Several common errors are specific to the Java Integration stage.

## Errors in the stage editor

When using the Java Integration stage, you might encounter certain GUI errors.

### Symptoms

- When you click **Configure** on the **Java Integration stage property editor**, you might encounter the following warning message:

  ```
  Failed to instantiate the resource wrapper class
  ```

- When you click **Select** on the stage property **Usage>User class** property, you might encounter the following warning message:

  ```
  Failed to send the request to the handler: The agent at HOST123:31531
  is not available.
  ```

### Resolving the problem

Verify that the ASB agent is working and restart the ASB agent if necessary.

The ASB agent log file can help you understand the errors that occur in the stage editor. The file is named `asb-agent-xx.out` and is in the `ASBNode/logs` folder if `com.ibm.iis.cas.level=ALL` is added into the `ASBNode/conf/asbagent-logging.properties` file.

## Runtime errors

After you configure and compile your Java Integration stage job, you might encounter certain run time errors.

### Stage configuration issues

When you run a job that includes a Java Integration stage, you get an error that is related to the configuration of the stage.

#### Symptoms

When a user class `userClass` is not found in the classpath that is specified at the **Java Integration stage property editor**, the following error might occur:

```
com.ascential.e2.common.CC_Exception: java.lang.ClassNotFoundException:
userClass
 at java.net.URLClassLoader.findClass(URLClassLoader.java:588)
 at java.lang.ClassLoader.loadClassHelper(ClassLoader.java:743)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:711)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:690)
 at com.ibm.is.cc.javastage.connector.CC_JavaConnection.connect
(CC_JavaConnection.java:155)
 at com.ibm.is.cc.javastage.connector.CC_JavaConnection.connect
(CC_JavaConnection.java: 250)
```

If a user class fails the connection test at the initialization time, the following error might occur for a class that is not inherited from `com.ibm.is.cc.javastage.api.Processor` class or `com.ascentialsoftware.jds.Stage` class.

```
Connection test failed (PXBridgeOp::negotiate, file pxbridge.c, line 1,684)
```

### Resolving the problem

Add the user class file that is inherited from the Processor or Stage class, or a `.jar` file that contains the class to a directory that the parallel engine can access. In the Java Integration stage, set the **Usage - Java - Classpath** property to the path name that contains the file.

To confirm that the correct class path was specified, click **Select** on the **Usage - User class** property and select the user class.

## JVM related issues

You might encounter an error related to a JVM creation when running a Java Integration stage.

### Symptoms

The conductor runs as a single process and creates a single JVM. Options for the JVM are specified in one of Java Integration stages in the job created. When you run a job which has multiple Java Integration stages, you might encounter the following error:

```
JVMDUMP039I Processing dump event
"systhrow", detail "java/lang/OutOfMemoryError"
at 2013/08/15 11:46:11
com.ibm.tools.attach.enable please wait.
JVMDUMP032I JVM requested System dump using
'C:\IBM\InformationServer\Server\Projects\dstage1\core.20130815.
114611.3932.0001.dmp'
in response to an event
```

This error might occur if maximum heap size is enough for one Java Integration stage but not all of the Java Integration stages.

### Resolving the problem

1. Verify which JVM options are required.
2. Set an environment variable **CC_MSG_LEVEL** to 2.
3. Compile and run the job.

   You can see the JVM options in the following section of the job log:

   ```
   CC_JNICommon.cpp:(634)
   The JVM is started with these options =
   -Dcom.ibm.tools.attach.enable=no
   -Dcom.ibm.is.cc.options=noisfjars
   -Xmx256m -Djava.class.path=
   C:\IBM\InformationServer\Server\DSEngine/../DSComponents/bin/ccapi.jar;.;
   C:\IBM\SQLLIB\java\db2java.zip;C:\IBM\SQLLIB\java\db2jcc.jar;
   C:\IBM\SQLLIB\java\sqlj.zip;C:\IBM\SQLLIB\java\db2jcc_license_cu.jar;
   C:\IBM\SQLLIB\bin;C:\IBM\SQLLIB\java\common.jar
   ```

4. You need to verify how much of 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`.

   ```
   $CC_JVM_OVERRIDE_OPTIONS =
   -Dcom.ibm.tools.attach.enable=no
   -Dcom.ibm.is.cc.options=noisfjars
   -Xmx512m -Djava.class.path=
   C:\IBM\InformationServer\Server\DSEngine/../DSComponents/bin/ccapi.jar;.;
   C:\IBM\SQLLIB\java\db2java.zip;C:\IBM\SQLLIB\java\db2jcc.jar;
   C:\IBM\SQLLIB\java\sqlj.zip;C:\IBM\SQLLIB\java\db2jcc_license_cu.jar;
   C:\IBM\SQLLIB\bin;C:\IBM\SQLLIB\java\common.jar
   ```

## Java Pack related issues

You might encounter a fatal error when running a Java Integration stage migrated from an existing Java Pack job.

### Symptoms

1. 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:588)
   at java.lang.ClassLoader.loadClass(ClassLoader.java:743)
   at java.lang.ClassLoader.loadClass(ClassLoader.java:711)
   at java.lang.ClassLoader.loadClass(ClassLoader.java:690)
   at com.ibm.is.cc.javastage.connector.CC_JavaConnection.connect
   (CC_JavaConnection.java:155)
   at (com.ibm.is.cc.javastage.connector.CC_JavaConnection::connect
   (CC_JavaConnection.java: 250)
   ```

2. You might encounter the following fatal error when running a Java Integration stage job using Java Pack API:

   ```
   JavaPackTransformer: java.lang.IllegalAccessError:
   Class com/ibm/is/cc/javastage/connector/CC_JavaPackProcessor
   illegally accessing "package private" member of class
   com/ascentialsoftware/jds/Stage
   at com.ibm.is.cc.javastage.connector.CC_JavaPackProcessor.<init>
   (CC_JavaPackProcessor.java: 121)
   at com.ibm.is.cc.javastage.connector.CC_JavaConnection.connect
   (CC_JavaConnection.java: 167)
   ```

### Resolving the problem

The first error might occur because in Java Pack, the `$DSHOME` directory is considered the root directory of the relative path specified at the stage property **User's Classpath** on the **Java Pack property editor**. In the Java Integration stage `$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 first 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>`.

The second 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 second 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**.

## Stage configuration issues

When you run a job that includes a Java Integration stage, you get an error that is related to the configuration of the stage.

### Symptoms

When a user class `userClass` is not found in the classpath that is specified at the **Java Integration stage property editor**, the following error might occur:

```
com.ascential.e2.common.CC_Exception: java.lang.ClassNotFoundException:
userClass
 at java.net.URLClassLoader.findClass(URLClassLoader.java:588)
 at java.lang.ClassLoader.loadClassHelper(ClassLoader.java:743)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:711)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:690)
 at com.ibm.is.cc.javastage.connector.CC_JavaConnection.connect
(CC_JavaConnection.java:155)
 at com.ibm.is.cc.javastage.connector.CC_JavaConnection.connect
(CC_JavaConnection.java: 250)
```

If a user class fails the connection test at the initialization time, the following error might occur for a class that is not inherited from `com.ibm.is.cc.javastage.api.Processor` class or `com.ascentialsoftware.jds.Stage` class.

```
Connection test failed (PXBridgeOp::negotiate, file pxbridge.c, line 1,684)
```

### Resolving the problem

Add the user class file that is inherited from the Processor or Stage class, or a `.jar` file that contains the class to a directory that the parallel engine can access. In the Java Integration stage, set the **Usage - Java - Classpath** property to the path name that contains the file.

To confirm that the correct class path was specified, click **Select** on the **Usage - User class** property and select the user class.

## Column mapping related issues

You might encounter a fatal error and warning when running a Java Integration stage job using JavaBeans.

### Symptoms

1. 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)
   ```

2. 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)
```

### Resolving the problem

This fatal 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.

The warning 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 fix the issues, perform the following actions:

1. Verify the link property **Usage - Column mapping**.
2. Click **Configure** to update the link property.
3. You can see the current setting on **Column Mapping Editor**. Modify the mapping by selecting the required setting from the pull down list of each DataStage column.
4. If you want to see all the JavaBean properties, click **Browse Objects**. You can see and select them by clicking check boxes on the **Select Bean Properties** dialog.
5. If you want to save new selections, click **OK** on the **Select Bean Properties** dialog and click **Finish** on the **Column Mapping Editor**.

## Link related issues

You might encounter a fatal error when running a Java Integration stage job with multiple input links or reject links.

### Symptoms

1. 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)
```

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

```
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)
```

### Resolving the problem

The first error might occur because you might have changed the link order after configuring the column mapping.

To fix the error, 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.

To fix the second error, 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 9. 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.

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. Because the information center uses XHTML, you can set display preferences in your browser. This 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:

**wsetsrc***[-S server] [-l label] [-n name] source*

The *source* argument is the only required argument for the **wsetsrc** 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:

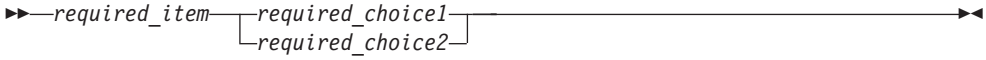
- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
  - The >>--- symbol indicates the beginning of a syntax diagram.
  - The ---> symbol indicates that the syntax diagram is continued on the next line.
  - The >--- symbol 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).


- Optional items appear below the main path.

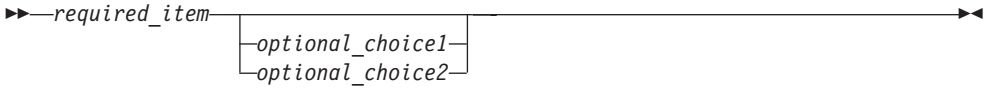

  If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.


- If you can choose from two or more items, they appear vertically, in a stack.

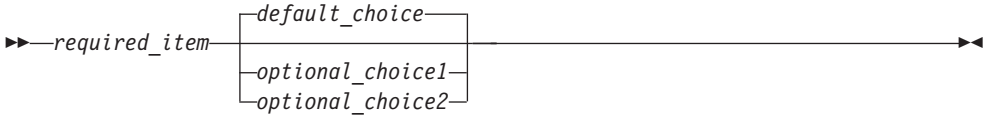
  If you must choose one of the items, one item of the stack appears on the main path.




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

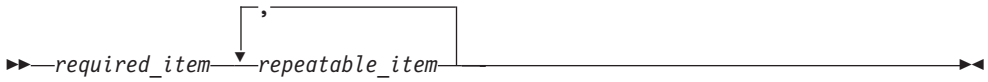



  If one of the items is the default, it appears above the main path, and the remaining choices are shown below.



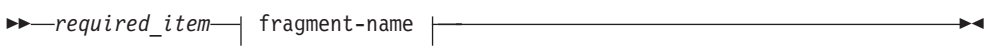

- An arrow returning to the left, above the main line, indicates an item that can be repeated.


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 5. IBM resources*

| Resource | Description and location |
|---|---|
| IBM Support Portal | You can customize support information by choosing the products and the topics that interest you at www.ibm.com/support/entry/portal/Software/Information_Management/InfoSphere_Information_Server |
| Software services | You can find information about software, IT, and business consulting services, on the solutions site at www.ibm.com/businesssolutions/ |
| My IBM | 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 www.ibm.com/account/ |
| Training and certification | 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 http://www.ibm.com/training |
| IBM representatives | You can contact an IBM representative to learn about solutions at www.ibm.com/connect/ibm/us/en/ |

# Appendix E. Accessing the product documentation

Documentation is provided in a variety of formats: in the online IBM Knowledge Center, in an optional locally installed information center, and as PDF books. You can access the online or locally installed help directly from the product client interfaces.

IBM Knowledge Center is the best place to find the most up-to-date information for InfoSphere Information Server. IBM Knowledge 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 IBM Knowledge Center from the installed product or from a web browser.

## Accessing IBM Knowledge Center

There are various ways to access the online documentation:

- Click the **Help** link in the upper right of the client interface.
- 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.
- Type the address in a web browser, for example, when you are not logged in to the product.

  Enter the following address to access all versions of InfoSphere Information Server documentation:

  `http://www.ibm.com/support/knowledgecenter/SSZJPZ/`

  If you want to access a particular topic, specify the version number with the product identifier, the documentation plug-in name, and the topic path in the URL. For example, the URL for the 11.3 version of this topic is as follows. (The ⇒ symbol indicates a line continuation):

  ```
  http://www.ibm.com/support/knowledgecenter/SSZJPZ_11.3.0/⇒
  com.ibm.swg.im.iis.common.doc/common/accessingiidoc.html
  ```

  **Tip:**

  The knowledge center has a short URL as well:

  `http://ibm.biz/knowctr`

  To specify a short URL to a specific product page, version, or topic, use a hash character (#) between the short URL and the product identifier. For example, the short URL to all the InfoSphere Information Server documentation is the following URL:

  `http://ibm.biz/knowctr#SSZJPZ/`

  And, the short URL to the topic above to create a slightly shorter URL is the following URL (The ⇒ symbol indicates a line continuation):

  ```
  http://ibm.biz/knowctr#SSZJPZ_11.3.0/com.ibm.swg.im.iis.common.doc/⇒
  common/accessingiidoc.html
  ```

## Changing help links to refer to locally installed documentation

IBM Knowledge Center contains the most up-to-date version of the documentation. However, you can install a local version of the documentation as an information center and configure your help links to point to it. A local information center is useful if your enterprise does not provide access to the internet.

Use the installation instructions that come with the information center installation package to install it on the computer of your choice. After you install and start the information center, you can use the **iisAdmin** command on the services tier computer to change the documentation location that the product F1 and help links refer to. (The ⇒ symbol indicates a line continuation):

**Windows**

```
IS_install_path\ASBServer\bin\iisAdmin.bat -set -key ⇒
com.ibm.iis.infocenter.url -value http://<host>:<port>/help/topic/
```

**AIX® Linux**

```
IS_install_path/ASBServer/bin/iisAdmin.sh -set -key ⇒
com.ibm.iis.infocenter.url -value http://<host>:<port>/help/topic/
```

Where <host> is the name of the computer where the information center is installed and <port> is the port number for the information center. The default port number is 8888. For example, on a computer named server1.example.com that uses the default port, the URL value would be http://server1.example.com:8888/help/topic/.

## Obtaining PDF and hardcopy documentation

- The 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.
- 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.

# Appendix F. Providing feedback on the product documentation

You can provide helpful feedback regarding IBM documentation.

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

- To provide a comment about a topic in IBM Knowledge Center that is hosted on the IBM website, sign in and add a comment by clicking **Add Comment** button at the bottom of the topic. Comments submitted this way are viewable by the public.
- To send a comment about the topic in IBM Knowledge Center to IBM that is not viewable by anyone else, sign in and click the **Feedback** link at the bottom of IBM Knowledge Center.
- Send your comments by using the online readers' comment form at 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. This material may be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

## 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.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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.

## Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session or persistent cookies. If a product or component is not listed, that product or component does not use cookies.

*Table 6. Use of cookies by InfoSphere Information Server products and components*

| Product module | Component or feature | Type of cookie that is used | Collect this data | Purpose of data | Disabling the cookies |
|---|---|---|---|---|---|
| Any (part of InfoSphere Information Server installation) | InfoSphere Information Server web console | • Session<br>• Persistent | User name | • Session management<br>• Authentication | Cannot be disabled |
| Any (part of InfoSphere Information Server installation) | InfoSphere Metadata Asset Manager | • Session<br>• Persistent | No personally identifiable information | • Session management<br>• Authentication<br>• Enhanced user usability<br>• Single sign-on configuration | Cannot be disabled |

*Table 6. Use of cookies by InfoSphere Information Server products and components (continued)*

| Product module | Component or feature | Type of cookie that is used | Collect this data | Purpose of data | Disabling the cookies |
|---|---|---|---|---|---|
| InfoSphere DataStage | Big Data File stage | • Session<br>• Persistent | • User name<br>• Digital signature<br>• Session ID | • Session management<br>• Authentication<br>• Single sign-on configuration | Cannot be disabled |
| InfoSphere DataStage | XML stage | Session | Internal identifiers | • Session management<br>• Authentication | Cannot be disabled |
| InfoSphere DataStage | IBM InfoSphere DataStage and QualityStage Operations Console | Session | No personally identifiable information | • Session management<br>• Authentication | Cannot be disabled |
| InfoSphere Data Click | InfoSphere Information Server web console | • Session<br>• Persistent | User name | • Session management<br>• Authentication | Cannot be disabled |
| InfoSphere Data Quality Console | | Session | No personally identifiable information | • Session management<br>• Authentication<br>• Single sign-on configuration | Cannot be disabled |
| InfoSphere QualityStage Standardization Rules Designer | InfoSphere Information Server web console | • Session<br>• Persistent | User name | • Session management<br>• Authentication | Cannot be disabled |
| InfoSphere Information Governance Catalog | | • Session<br>• Persistent | • User name<br>• Internal identifiers<br>• State of the tree | • Session management<br>• Authentication<br>• Single sign-on configuration | Cannot be disabled |
| InfoSphere Information Analyzer | Data Rules stage in the InfoSphere DataStage and QualityStage Designer client | Session | Session ID | Session management | Cannot be disabled |

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

## 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.

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, LACS$^{Link}$, ZIP, ZIP + 4, ZIP Code, Post Office, Postal Service, USPS and United States Postal Service. IBM Corporation is a non-exclusive DPV and LACS$^{Link}$ licensee of the United States Postal Service.

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

# Index

Printed in USA

SC19-4336-00