IBM InfoSphere Information Server
Version 11 Release 3

Connectivity Guide for IBM WebSphere ILOG JRules

SC19-4338-00

IBM InfoSphere Information Server
Version 11 Release 3

Connectivity Guide for IBM WebSphere ILOG JRules

SC19-4338-00

Note

Before using this information and the product that it supports, read the information in "Notices and trademarks" on page 43.

Contents

IBM WebSphere ILOG JRules

With the IBM® WebSphere® ILOG® JRules Connector, you can invoke ILOG JRules rulesets from within a job.

Note: The products in the IBM WebSphere ILOG JRules Business Rules Management System (BRMS) family were rebranded as IBM Operational Decision Manager. The component that the connector accesses is now called IBM Decision Server. The InfoSphere® Information Server clients and documentation are not updated to reflect this change.

Overview

ILOG JRules Business Rules Management System (BRMS) allows customers to externalize complex business rules from applications. With the ILOG JRules stage, you can invoke complex business rules within the context of a job.

The stage provides DataStage parallel jobs with access to the ILOG JRules engine. The JRules engine can be used to perform complex data transformations on the records in the job.

If the connector is configured to access JRules in core engine mode, it invokes the rulesets that reside in the ruleset archive files on the IBM InfoSphere Information Server engine host. If the connector is configured to access JRules in locally managed (J2SE) or remotely managed (J2EE) mode, it invokes the rulesets that are persisted in the ruleset repository.

ILOG JRules connector consists of a design time component and a runtime component.

The design time component consists of the stage editor that collects inputs from the user. It also consists of the configuration wizard that is used to configure the stage properties and link schema definitions of the connector based on the parameter definitions of the specified ruleset.

The runtime component of the connector is responsible for invoking the rulesets within the context of the DataStage job. ILOG JRules accepts input data in the form of Java objects and the output data returned by ILOG JRules also consists of Java objects. The basic functionality of the runtime component is to read the input DataStage records sent by the upstream stage and convert them into Java objects. It then invokes the ruleset by using the instantiated Java objects and converts the Java objects returned by the ruleset back into DataStage records. These records are then sent to the downstream stage. Consequently, the ILOG JRules stage can operate only in the request/reply mode, which further means that it must be positioned in the job as a midstream stage and not the first or the last stage. It is similar, for example, to the use of the Transformer stage.

JRules engine mode

The JRules engine mode specifies the type of interface that the connector uses to locate the ruleset and configure the stage at design time and to establish the JRules session and invoke the ruleset at runtime. You can select Core engine, J2SE RES XU, or J2EE RES XU as the JRules engine mode.

Core engine

In this engine mode, the connector accesses the JRules engine directly through the engine JAR file. The connector points the JRules engine to the physical ruleset archive file that contains the definition of the ruleset to be executed.

To use this mode, you must export the ruleset as a ruleset archive. The ruleset is exported as a JAR file that must be copied on the machine on which IBM InfoSphere Information Server engine is running. The path of this JAR file must be specified in the **Ruleset location** stage property.

J2SE RES XU mode

In this engine mode, the JRules engine runs as a locally managed Rule Execution Server (RES) in the same Java process as the connector. To invoke the ruleset, the connector points the JRules engine to the ruleset definition in the ruleset repository.

The ruleset repository can either be a file-based repository or a database-based repository. If the repository is file-based, then it must be on the same machine on which IBM InfoSphere Information Server engine is running. The configuration on whether to use a file-based repository or a database-based repository is specified in the `ra.xml` JRules configuration file. For more details on this configuration file, please refer to the JRules documentation. When using the Java 2 Standard Edition (J2SE) RES XU the **Ruleset location** stage property specifies the path of the ruleset in the repository.

J2EE RES XU mode

In this engine mode, the JRules engine runs as a remotely managed RES on the Java 2 Enterprise Edition (J2EE) server. To invoke the ruleset, the connector points the JRules engine to the ruleset definition in the ruleset repository. The connection to the ruleset repository is configured when the RES is deployed to the J2EE server. During job execution, the connector acts as the remote EJB3 client and establishes connection to the EJB3 rule session component that must be deployed to the J2EE server on which the Rule Execution Server is deployed. When using the J2EE RES XU the ruleset location stage property specifies the path of the ruleset in the repository.

Note: For batch types of operations, the Core engine and J2SE RES XU modes generally provide for better performance than the J2EE RES XU mode because the connector exchanges Java objects with the JRules engine that runs in the same Java Virtual Machine as the connector. In the J2EE RES XU engine mode, the connector accesses the JRules engine that runs on the J2EE server through the remote EJB interface and the Java objects that they exchange need to be serialized and deserialized as part of the process which can incur a significant overhead.

Batch mode and key mode processing

Each input link of the JRules Connector stage corresponds to either an IN or an IN_OUT parameter of the ruleset specified for the stage. The connector converts records retrieved on the input links to Java values and passes them to JRules as the values of the corresponding IN and IN_OUT ruleset parameters. A key concept associated with the run time functionality is that of rule execution cycle. During each rule execution cycle the connector reads one or more records from the input links, converts them to Java objects and sends them to JRules.

The number of records retrieved from the input links during each rule execution cycle is governed by batch mode and key mode. You can configure either batch mode or key mode of processing records. The strategy that the connector uses to process records from the first input link is different from the strategy it uses to process records from the remaining input links. The connector internally classifies input links into master and secondary links. The first input link is the master link, and the remaining input links are secondary links.

The information in this topic applies to scenarios in which the following conditions are met:

- None of the input links are marked as buffer links. For information about the use of batch and key modes with buffer links, refer to the topic *Buffer links.*
- The XOM type for the ruleset parameters is Java XOM. For information about the use of batch and key modes with Dynamic XOM, refer to the topic *Execution Object Model types.*

Record processing strategy

The connector explicitly sets values for all IN and IN_OUT parameters of the ruleset before invoking the ruleset. In some cases the value that is set for a parameter can be the NULL value. But the connector does not invoke a ruleset without previously setting values for all IN and IN_OUT parameters of the ruleset.

When no records are available on the master link, the connector ensures that no records are available on any of the secondary links and the job completes successfully. However, if there are more records available for some secondary link, the connector rejects them if the reject link is defined for that secondary link and the **Leftover record error** option is selected for the reject link. Otherwise, the connector logs an error and the job fails.

If records are not available on a secondary link, the action that the connector takes depends on whether ruleset parameter associated with that link is of a Java array type or not. If it is, the connector creates an empty array and sets it as the ruleset parameter value. Otherwise, the connector treats the condition as an error and rejects the records that it received on all the input links (including the master link), during the current rule execution cycle, but only if the reject links are defined for those input links and the **Leftover record error** option was selected for the reject links. Otherwise, the connector logs an error and the job fails. The additional rules that apply to the record processing strategy depend on the mode in which the stage is configured to run.

Batch mode

To enable the batch mode, the **Enable batch processing** stage property must be set to **Yes** and the **Enable key processing** stage property must be set to **No**. A non-negative integer value *n* also must be specified for the **Batch size** property. The default value is 1. In the batch mode, for each rule execution cycle, the connector reads all available records from each of the input links, but not more than *n* records per link, where *n* is the batch size specified for the stage. The batch size of 0 means unlimited.

When the connector retrieves the necessary number of records on all input links, it converts them to Java objects which it then sets as the ruleset parameter values and invokes the ruleset. If the ruleset executes successfully, the connector produces records on the output links based on the OUT and IN_OUT ruleset parameter

values, and repeats the process for the next ruleset execution. If the ruleset execution fails, the connector logs an error and the job fails.

Consider an example of a job where the ILOG JRules stage has two input links, with each link containing only one column of type Integer. Let the data received on the input links be specified as follows (the data on the right arrives first):

Link 1

19 17 16 15 14 12 11 10

Link 2

20 18 17 15 12 10 8

Let batch mode be enabled and let the batch size be 2. For this data, there are four rule execution cycles. The data sent in each rule execution cycle is as follows:

Rule execution cycle 1

Link 1: 11 10

Link 2: 10 8

Rule execution cycle 2

Link 1: 14 12

Link 2: 15 12

Rule execution cycle 3

Link 1: 16 15

Link 2: 18 17

Rule execution cycle 4

Link 1:19 17

Link 2: 20

Key mode

To enable the key mode, the **Enable key processing** stage property must be set to **Yes**. You must select the input link columns to serve as the key columns by using the **Key column[*n*]** property, where *n* is a positive integer and represents the index of the key column. The indexes start at 1 for the first selected key column and increment as more columns are added. The name of each key column to be used as the key column must be specified in the **Column name** stage property under the corresponding **Key column[*n*]** property. At least one key column must be specified, which means that at least **Key column[1]** property must be present and its **Column name** child property value must be set. To add a **Key column[2]**, right-click **Key column[1]** and select **Add property value**. Repeat the process to add more **Key column[*n*]** values. To set the **Column name** stage property for a

particular **Key column[*n*]** it is possible to click the **Select** button displayed on the right side of **Column name**. The columns that are offered for selection are the columns that exist on all input links.

For each selected key column, it is also necessary to specify the direction in which the records on the input links are sorted in respect to the values of that particular key column. This is specified in the **Sort direction** option under the corresponding **Key column[*n*]**. The supported values are **Ascending** and **Descending**. The default value is **Ascending**. When the connector compares two key column values to determine their position relative to each other, it compares the Java values created from those column values. For key columns that contain character values (for example VarChar columns), the comparing of values is done in a case-sensitive manner. When the connector compares NULL value to non-NULL value, it considers the NULL value to be smaller than the non-NULL value. For a nullable key column and the ascending sort order the records with NULL values for that key column appear before the records with non-NULL values for that same key column.

When running in key mode, the connector automatically enforces at runtime the sorting of records on its input links based on the key column names and sort directions specified in the stage. Also, if the connector stage is configured to run in key mode, in parallel, on more than one node, and if the **Partition type** field on the **Partitioning** tab for any input link is set to the default **(Auto)** value, the connector enforces hash partitioning of the records on that input link and for the hash key uses the key columns specified in the stage. This ensures that the records with the matching key column values are processed by the same stage instance (running on the same node).

The batch size specified for the stage in the **Batch size** property plays an important role in the key mode. If the **Enable batch processing** property is set to **No** when **Enable key processing** is set to **Yes**, the connector runs in key mode and assumes the batch size of 0 (unlimited). When the connector reads records from the input links in this mode, in addition to the batch size it also takes into account the values of the key columns present in the records. For each ruleset execution the connector reads up to *n* records from the master link where *n* represents the batch size specified for the stage. If the batch size *n* is different from 0, the connector reads up to *n* records from the link regardless of the key values in those records. However, if the batch size is 0 (unlimited), the connector reads all available records from the master link that have matching key column values.

After it reads the records from the master link, the connector proceeds to read records from the remaining secondary links which have key column values that match key column values of any of the records retrieved on the master link for that same rule execution cycle. The number of records that the connector reads from the secondary link depends on whether the ruleset parameter for that link is based on a Java array type or not. If the ruleset parameter is based on an array type, the connector reads all available matching records and uses them to prepare the value for the ruleset parameter. If the ruleset parameter is not based on an array type, the connector reads only one matching record from the link. The connector then invokes the ruleset.

The connector then checks if, for any of the parameters not based on array types there exists another matching record on the corresponding secondary input link. If such a record exists, the connector uses it for the ruleset parameter value and invokes the ruleset again. When invoking the ruleset again, the connector reuses the value from the previous ruleset execution for each ruleset parameter for which

it did not read any new matching records for the current ruleset execution. The connector repeats this process until it cannot find any more new matching records for any of the ruleset parameters. It then proceeds to read new records from the master link in preparation for the next ruleset execution.

Consider an example of a job where the ILOG JRules Stage has two input links and each link has two columns. In Link 1, first column is of type Integer and the second column is of type VarChar(5). In Link 2, the first column is of type Integer and the second column is of type Double. Let the data received on the input links be as follows (the data on the right arrives first):

Link 1

(16, Sam) (16, Sam) (16, Sam) (12, John) (12, John) (10, Rick) (10, Rick) (10, Rick)

Link 2

(16, 32.4) (12, 32.22) (12, 24.22) (12, 53.33) (12, 23.233) (10, 32.55) (10, 25.77)

Let key mode be enabled and let the batch size be 0. The first column of type Integer is the key column and the sort order for the key is **Ascending**. Let the ruleset parameters associated with the input links be of Java array types. Then for this data there are three rule execution cycles. The data sent in each rule execution cycle is as follows:

Rule execution cycle 1

Link 1: (10, Rick) (10, Rick) (10, Rick)

Link 2: (10, 32.55) (10, 25.77)

Rule execution cycle 2

Link 1: (12, John) (12, John)

Link 2: (12, 32.22) (12, 24.22) (12, 53.33) (12, 23.233)

Rule execution cycle 3

Link 1: (16,Sam) (16,Sam) (16,Sam)

Link 2: (16, 32.4)

If for some of the secondary links the connector detects a record with the key column values that do not match the key column values in any of the master records, and if no records with the matching key column values were previously retrieved on the same link for the same ruleset execution, the connector specifies an empty array for the ruleset parameter value for that link. The empty array is specified only if that ruleset parameter is of a Java array type and only if based on the sorting order specified for the column keys the connector determines that there is a possibility for the key column values in the detected record to match the key column values in a record that the connector is yet to retrieve on the master link.

Otherwise, the connector attempts to reject the records that caused the key mismatch error. The connector compares the key column values of the detected record on the current secondary link and the key column values of the records that

it already retrieved on the master link for the current ruleset execution. When performing this comparison the connector takes into account the sort direction that was specified for the key columns. If it is possible for the next record on the current secondary link to have the key column values that match the key column values of any record that was already retrieved on the master link, the connector rejects the record that it read on the current secondary link. Otherwise, it rejects the records that it previously retrieved on the remaining input links, including the records from the master input link.

Consider an example of a job where the ILOG JRules stage has three input links. Let the key column on the three links be C1 of type Integer. Let the value of column C1 on the three links be as follows (data on the right arrives first):

Link 1 (Master): 5 5 5

Link 2: 5 5

Link 3: 6

Let the batch size be set to 0.

The behavior of the connector under various operating modes is as follows:

1. The ruleset parameters for all the input links are of array type and the sorting order for the column C1 is **Ascending**. The connector invokes the ruleset and passes three records with C1 value of 5 for the first ruleset parameter, two records with C1 value of 5 for the second ruleset parameter, and 0 records for the third ruleset parameter. The connector determines based on the sort order of column C1 that it is still possible that the next record on the master link has the C1 value of 6 and that presents a match for the current secondary link record and hence the record on Link 3 is not rejected.
2. The ruleset parameter for Link 3 is not of Java array type and the sorting order is **Ascending**. In this case, the connector rejects the records with the C1 value of 5 that it already retrieved on the first and second input links, because the records that potentially arrive next on the third input link after the record with C1 value of 6 can have only the C1 value of 6 or greater, so none of them are able to match the C1 value of 5.
3. The ruleset parameter for Link 3 is not of Java array type and the sorting order is **Descending**. In this case, the connector rejects the record with the C1 value 6 that it retrieved on the third input link because there is a possibility that the next record on this link has the C1 value of 5 and therefore match the records retrieved on the first two input links.

After the connector rejects the records due to the key mismatch, it keeps trying to prepare ruleset parameter values for the current ruleset execution, by reading records from the first input link from which it rejected the records. This is either the master link, in which case the connector start preparing ruleset parameter values from scratch, or the current secondary link in which case the connector proceeds to read records on that link to try to match them with the records it already received on the master link for the current ruleset execution.

If a record must be rejected due to key mismatch, the connector can do that only if the record to be rejected came from an input link for which the reject link is defined and only if the **Key mismatch error** option is selected on that reject link. Otherwise, the connector logs an error and the job fails.

In the batch mode, the connector enforces the batch size value of 1 if the ruleset parameter for any of the input links is not based on a Java array type. In the key mode, the connector enforces the batch size value of 1 if the ruleset parameter for the master link is not based on a Java array type. If at runtime the connector needs to override the **Batch size** value specified by the user and enforce the value of 1, it writes an informational message in the job log about this change.

If the records arrive on the input links of the stage in transaction waves, the connector processes records in each wave independently from the records in another wave. The records from one wave are not combined with the records from another wave in order to produce ruleset parameter values for a single ruleset execution.

Execution Object Model types

ILOG JRules connector can use either the Dynamic Execution Object Model (Dynamic XOM) or Java Execution Object Model (Java XOM). The connector does not support ruleset parameters with a combination of the Dynamic XOM and Java XOM types. All the parameters of the ruleset must be based of the same XOM type. To ensure this, the **XOM type** property is associated with the stage and not the link.

Java XOM

To configure the stage for Java XOM mode, the **XOM type** stage property must be set to **Java XOM**. If the configuration wizard is used to import ruleset parameters and configure the stage for a ruleset based on the Java XOM, the wizard automatically sets **XOM type** property on the stage to **Java XOM**.

When the **Java XOM** option is selected, the **Ruleset parameter class** link property value must be specified. The specified value represents the Java type of the ruleset parameter associated with that link. If the specified type is one of the basic Java types supported by the connector, a single column on the link corresponds to the entire ruleset parameter. Otherwise, the columns on the input links correspond to the arguments of the methods of the Java class of ruleset parameter and columns on the output links correspond to the return values of the methods of the ruleset parameter Java class. For information about basic Java types in the context of the connector please refer to the topic *Basic Java types and methods.*

Dynamic XOM

The connector supports invoking rulesets that are based on the Dynamic XOM. In this mode, the ruleset parameter values are XML documents. The XML documents must be valid in respect to the XML schema documents that were used to define the XOM for the ruleset.

To configure the stage for Dynamic XOM mode, the **XOM type** stage property must be set to **Dynamic XOM**. If the configuration wizard is used to import ruleset parameters and configure the stage for a ruleset based on the Dynamic XOM, the wizard automatically sets **XOM type** property on the stage to **Dynamic XOM**.

When the stage is configured for Dynamic XOM, the connector reads one record from each input link, produces XML documents from those records, and invokes the ruleset and passes the produced XML documents as the values for IN and IN_OUT parameters of the ruleset.

When the connector reads a record from an input link, it searches for the field in the record that contains the XML data from which to create the XML document value for the ruleset parameter. The entire XML document must be stored in a single field of the record. The remaining fields of the record can be used as key columns values to correlate records across multiple input links, or they can be fields that the connector propagates from input links to output links.

The column on the input link to be used for the XML documents for ruleset parameters must be identified so that the connector has access to the field in each retrieved record on that link which contains the actual XML data to be used as the ruleset parameter value. The identified column must be of a character-based DataStage data type: Char, VarChar, LongVarChar, NChar, NVarChar, and LongNVarChar. If there are multiple such columns, only one of them must have the **Description** attribute value that includes the `CC_JRules();` expression. The **Description** attribute is present in the columns tab which is visible after clicking any of the input or output links in the stage. If, after checking these conditions there are zero or more than one eligible columns on the link, the job fails with an error.

The batch mode is not supported when the connector is configured to invoke ruleset based on Dynamic XOM. The batch size of 1 is implicitly used in this case. The connector does not support passing multiple XML documents in a single ruleset parameter value.

The key mode is supported by the connector configured to invoke ruleset based on the Dynamic XOM. In this case, the specified key columns can be used for sorting and partitioning of input records across multiple input links and multiple processing nodes of the stage, to guarantee that the connector is always invoking the ruleset with the correlated XML document values across all IN and IN_OUT parameters of the ruleset.

DataStage fields and ruleset parameters mapping

The ILOG JRules connector converts the records available in the input link into Java objects. The connector must understand the mapping between the connector stage fields and the ruleset parameters, in order to complete the conversion.

The mapping can also be identified automatically by using the configuration wizard. The following sections describe how the InfoSphere DataStage® fields are mapped to the ruleset parameters.

Java XOM

For Java XOM, the mapping between the fields on a link and the ruleset parameter associated with that link is determined based on the field attributes **Column name** and **SQL Type** and one of the following:

- The methods present in the ruleset parameter class
- The ruleset parameter name and data type (if it is of basic Java type).

For more information about basic Java types, refer to the topic *Basic Java types and methods.* If the ruleset parameter conforms to the conditions specified in the *Automatic mapping* section, then the mapping is done automatically based on the **Column name** attribute of the field. Otherwise, the mapping is done by using a mapping specification provided in the **Description** attribute of the field.

Automatic mapping

ILOG JRules connector can automatically identify the mapping between the link fields and the ruleset parameter if any of the following conditions are met:

1. If the ruleset parameter is of one of the basic Java types supported by the connector, then the **Column name** must correspond to the name of the ruleset parameter. The **SQL Type** of the field must be compatible with the basic Java type of the ruleset parameter. For more information about mapping or compatibility of data types, refer to topics *DataStage type to Java type mapping* and *Java type to DataStage type mapping*. In this case, the field is directly mapped to the ruleset parameter.
2. If the ruleset parameter is not of a basic Java type and the link is an input link, then the ruleset parameter class must have a method named **`setFieldName()`**, where **`FieldName`** is the **Column name** of the field. The comparison of the **Column name** and the method name is not case-sensitive. The method must not have any return type and must have a single argument and the **SQL type** of the field on the input link must be compatible with the data type of the argument expected by the method. For example the field **`firstName`** of type **`VarChar(10)`** on an input link maps to the first and only argument of the method **`void setFirstName(String name)`**. In this case, the input field is mapped to the argument expected by the method.
3. If the ruleset parameter is not of a basic Java type and the link is an output link, then the ruleset parameter class must have a method named **`getFieldName()`**, where **`FieldName`** is the **Column name** of the field. The comparison of the **Column name** and the method name is not case-sensitive. A special case is when the method is of **`boolean`** or **`java.lang.Boolean`** return type, and in that case the method is expected to be named **`isFieldName()`**. The return value from the method must be compatible with the **SQL type** of the field present on the output link. In this case, the field is mapped to the return value of the method.

If there is a field on the input link for which there is no method in the ruleset parameter class conforming to the second condition (and the **Description** attribute is empty) then the field values are not passed to ILOG JRules. Similarly, if there is a field in the output link for which there is no method in the ruleset parameter class conforming to the third condition (and the **Description** attribute is empty) then the field value is not set by the ILOG JRules connector.

Mapping using Description specification

If the ruleset parameter does not follow the conditions for automatic mapping, then the ILOG JRules connector provides a mechanism to specify the method to be invoked for setting or getting each field value in the InfoSphere DataStage input and output records. This is specified in the **Description** attribute of the schema fields on the **Columns** tab of the stage editor. For every field in the link schema, the **Description** carries information about the method to be invoked on the Java class associated with the link, to set, or get that field value. If no information is present in the **Description** attribute for any of the fields, then the ruleset parameter is assumed to conform to the specification mentioned in the *Automatic mapping* section.

For input link columns, the value of their **Description** attribute consists of one more concatenated expression, where each expression is in the `CC_JRules(methodName, parameterIndex);` format. The `methodName` specifies the name of the method to be invoked on the ruleset parameter object for this column. The `parameterIndex` value is an integer greater than or equal to 1 which specifies the index of the parameter (for the method) to which the column maps to.

For example, a Java object has a method `setFullName(String firstName, String lastName)` and the DataStage record contains two fields, *firstName* and *lastName*. In this case, the **Description** attribute for *firstName* is `CC_JRules(setFullName,1);` and that for *lastName* is `CC_JRules(setFullName,2);`. The *firstName* field value from a DataStage record is used for the first argument of `setFullName` method and the *lastName* field value from the same record is used for the second argument.

The *methodName* can also be a constructor. If the *firstName* column in the example is also used for the first argument of the constructor for the class `Customer`, then the **Description** attribute for the *firstName* is `CC_JRules(setFullName,1);CC_JRules(Customer,1);`.

For the output links, the method is called to get the value of the field from the Java object. Hence, the field does not expect any parameter. As a result, for fields on the output links, the parameter index is not required. The syntax of the **Description** attribute for the fields on the output link is: `CC_JRules(methodName);`. For each field, there must be only one method that can be called. If more than one method exists, then the ILOG JRules connector logs an error and the job fails.

When a single column on the link represents the entire ruleset parameter value, the **Description** attribute for the column has the `CC_JRules();` value. This is the case when the ruleset parameter is of one of the basic Java types supported by the connector.

Dynamic XOM

If a InfoSphere DataStage record field contains an XML document which is to be sent to the ruleset based on the Dynamic XOM, then the **Description** attribute for the field can contain the `CC_JRules();` value. When there are multiple fields in the InfoSphere DataStage record, the value is necessary to specify which of them contains the XML document.

If there is a single field in the record, then that field is automatically assumed to be the field containing the XML document and the **Description** attribute of the field can be left empty. The field that contains the XML document must be any of the InfoSphere DataStage character-based data types such as Char, NChar, VarChar, NVarChar, LongVarChar or LongNVarChar.

Field propagation strategies

ILOG JRules connector supports passing through unmatched fields from the input links to the output links. Unmatched fields are input link fields that are not used by the connector to invoke the ruleset. When you enable the propagation of unmatched fields, then the unmatched fields are propagated to the output links associated with the input links.

The connector automatically establishes the association between each input link that has the unmatched fields and the output links to which those fields must be propagated. For every unmatched field on an input link and for every output link associated with that input link, the connector checks if the output link contains the field with the same name as the unmatched field.

- If it does and if the output field is not mapped to any field in the XOM, then the field values of the input column are propagated and set as the field values of the output column.
- If it does not, the connector checks if the **Runtime column propagation** check box is selected for that output link. If it is selected, the connector adds the input

field definition to that output link schema and propagates the field values as the job runs. If the check box is not selected, the propagation of the input field definition and values does not take place.

You can use one of the following strategies to propagate unmatched fields:

Matching based on the order of records

When this strategy is selected, the connector propagates unmatched fields from the first input link to the first output link, from the second input link to the second output link, and so on. If there are more input links than the output links, then the field propagation is not performed for the input links for which there are no corresponding output links. An input link can be associated with only one output link. Similarly, an output link can be associated with only one input link.

When this strategy is used, the number of records on the input link that were used to invoke the ruleset must match the number of records produced on the associated output link after the ruleset executes. If the numbers do not match, the connector logs an error and the job fails.

Matching based on the propagation key column value

When this strategy is used, the connector propagates unmatched fields from an input link to an output link with the same propagation key column. The propagation key column is specified for each input link separately, through the **Propagation key column** link property.

The connector checks each input link to verify if there is a column marked as the propagation key column. If input links with column marked as propagation key column are found, and if the links have fields which are not used for ruleset execution, those fields are eligible for field propagation.

A propagation key column cannot be a column that is eligible for field propagation at the same time. If this scenario occurs, the job fails with an error. The connector inspects the input links and for each input link that has propagation key column as well as the columns eligible for propagation, the connector performs the following steps:

- It inspects all the output links that are not already associated with any of the previous input links.
- The output link that includes column with the same name and the data type as the propagation key column of the input link becomes associated with that input link.

With this strategy an input link can be associated with multiple output links, but an output link can be associated with only one input link.

When the connector reads records on an input link with the propagation key column enabled, it checks the propagation key column value of the record. If the value matches the value of another record retrieved on the same link for the current ruleset execution, the job fails with an error.

For each record that is sent to an output link, the record field is populated from the value of the respective OUT or IN_OUT parameter of the ruleset. The connector then checks if the output link is associated with an input link for field propagation. If it is associated, the connector searches for the input record that it

retrieved from the input link for the current ruleset execution and that has the same propagation key column value as the output record. If such a record exists, the connector copies the values of the propagated fields from the input record to the output record. If such an input record does not exist, the propagated fields in the output record remain unset.

Consider an example of a job where the ILOG JRules stage has two input links and two output links.

The links contain the columns as follows:

Input links

Link 1: (id, balance)

Link 2: (id, name, address)

Output links

Link 1: (name, cust_type, address)

Link 2: (name, outstanding, address)

If field propagation is enabled using the propagation key column strategy and the column **name** is specified as the propagation key column for Link 2, the connector detects that the **name** column is also defined on the two output links. If the **address** column is not being mapped to the ruleset in both the input as well as the output link, then the data of the **address** column from input Link 2 is sent to the **address** column of both the output links. The mapping is done based on the **name** field. For example, if input Link 2 has the following data:

1, "John", "123, Las Vegas Dr, Las Vegas, NV"

2, "Paul", "abc Drive, New York"

Records on the output links show up as follows:

Link 1

"John", "Gold", "123, Las Vegas Dr, Las Vegas, NV"

"Paul", "Silver", "abc Drive, New York"

Link 2

"John", "244.33", "123, Las Vegas Dr, Las Vegas, NV"

"Paul", "435.6445", "abc Drive, New York"

"Manish", 4363.453", ""

The data from the input **address** column is mapped to the output **address** column by using the value of the **name** column. The **name** field value in the third record on output Link 2 does not match the **name** field value in any of the records on input Link 2, so the **address** field value in the output record remains unset.

Matching based on the Java Object identifiers

When this strategy is used, the connector propagates unmatched fields from an input record to the output record that corresponds to the same Java object. For each input link the connector checks the Java type of the ruleset parameter specified for the link. If this type is not one of the basic Java types supported by the connector, the connector makes note of any columns on the link that are eligible for field propagation, meaning they are not used for producing the ruleset parameter values.

For each input link which has some columns (fields) eligible for propagation, the connector internally establishes association between that input link and the output links as follows:

- It inspects all the output links that are not already associated with any of the previous input links.
- The output links whose ruleset parameters are of the compatible Java types with the one specified for the input link become associated with the input link. Two Java types are compatible in this context if they are identical or if one of them is derived from the other. The Java array types are treated as the non-array types when checking for compatibility. For example if one Java type is ClassA and the other Java type is ClassB[], the two Java types are compatible if ClassA and ClassB are either identical or if one is derived from the other.

With this field propagation strategy in place, one input link can be associated with multiple output links, but one output link can be associated with only one input link.

For each record that the connector sends to an output link, it first populates the record fields from the value of the respective OUT or IN_OUT parameter of the ruleset and then checks if that output link is associated with an input link for the field propagation purposes. If it is associated, then the connector looks for the record retrieved from the input link for the current ruleset execution and that corresponds to the same Java object as the output record. If such a record exists, the connector copies the values for the propagated fields from the input record to the output record. If such a record does not exist, the propagated fields in the output record remain unset.

Java class path configuration

ILOG JRules Connector runs on the IBM InfoSphere Information Server engine tier host. ILOG JRules connector is a Java based connector and it requires access to various ILOG JRules resources such as ILOG JRules JAR files, Java Execution Object Model (XOM) classes, and the configuration files to function properly.

You must specify the resources required by the connector in the Java class path for the connector. You can specify the required resources in the **Classpath** property on the ILOG JRules connector stage page.

You can define a InfoSphere DataStage project environment variable to list all the necessary resources, and then add it to the job as job parameter specified as the **Classpath** property value. When the resources are specified in the **Classpath** property, they are made available to the connector during the job execution as well as during the configuration wizard session.

The use of the system Java class path to list the necessary resources is not recommended. If specified in the system class path, the resources might interfere with other Java processes that are running on the same machine. It is also not convenient to add resources to the system class path incrementally because it requires restarting the IBM InfoSphere DataStage engine in order to recognize the new entries added to the system class path. If the engine is a Windows system, then it requires a reboot. An example where the system class path needs to be updated periodically is to add new Java XOM JAR files for the connector to use to invoke new rulesets.

If it is not convenient to add the necessary resources in the **Classpath** property of each ILOG JRules stage, then you can do one of the following instead of setting the system class path value:

- For the runtime execution of the job, define the CLASSPATH environment variable at the IBM InfoSphere DataStage project level, and list the necessary resources in the default value for this environment variable. The specified CLASSPATH value is then picked up by every job in that IBM InfoSphere DataStage project. To override the default value specified at the project level, the jobs can import this environment variable as a job parameter and set its value to be used by the connector class path value for that particular job.
- For the configuration wizard, copy the necessary resources to *IS_HOME*/ASBNode/lib/java, where *IS_HOME* represents the home directory of the IBM InfoSphere Information Server installation on the IBM InfoSphere Information Server engine tier host.

  Note: You cannot add resources to the system class path for the configuration wizard sessions because these sessions run under the control of ASB Agent service. ASB Agent service is initialized with the internal custom classpath which does not include the resources specified in the system class path value.

Depending on the engine mode selected, you must have the specified JRules JAR files in the class path during the job execution as well as for the configuration wizard use.

The configuration wizard is running under ASB Agent service process on the IBM InfoSphere Information Server engine host. When the wizard loads Java XOM classes from the Java XOM JAR file specified in the **Classpath** property, that JAR file becomes locked during the wizard session. The wizard loads the XOM classes in each session independently, so if necessary it is generally possible to overwrite the XOM JAR between the wizard sessions, for example to put a newer version of the JAR in place of the old one. However, on some platforms the JAR file may remain locked even after the wizard session completes, because the ASB Agent process under which the classes were loaded is still running. In those cases, even if it is not possible to delete the locked JAR or move another JAR to replace of the locked JAR, it might still be possible to copy another JAR over the locked JAR. If that still does not work, another option is to place the new JAR to a new location and specify that location in place of the old one in the **Classpath** property. As the last option, the ASB Agent process can be restarted.

Core engine mode

You must have the *JRULES_HOME*/executionserver/lib/jrules-engine.jar JAR file in the class path during the job execution expression where, *JRULES_HOME* represents the ILOG JRules product home directory on the IBM InfoSphere Information Server engine host.

This JAR file is not required for the configuration wizard sessions.

When the ruleset for which the connector is configured is based on Java XOM, then the XOM classes of the ruleset parameters also need to be included in the class path. This applies to the job runtime as well as the configuration wizard sessions.

When the connector is configured for core engine mode, it does not require access to any configuration files.

J2SE RES XU mode

The following JRules JAR files need to be present in the class path during the job execution as well as for the configuration wizard use:

JRULES_HOME/executionserver/lib/jrules-engine.jar

JRULES_HOME/executionserver/lib/jrules-res-session-java.jar

JRULES_HOME/executionserver/lib/jrules-res-execution.jar

JRULES_HOME/executionserver/lib/sam.jar

JRULES_HOME/executionserver/lib/log4j-1.2.8.jar

JRULES_HOME/executionserver/lib/j2ee_connector-1_5-fr.jar

Where, *JRULES_HOME* represents the ILOG JRules product home directory on the IBM InfoSphere Information Server engine host.

When the ruleset for which the connector is configured is based on Java XOM, then the XOM classes of the ruleset parameters also need to be included in the class path. This applies to the job runtime as well as the configuration wizard sessions.

When the ruleset repository is JDBC-based, the JDBC driver implementation necessary for JRules engine to access the repository needs to be present in the class path. For example, if the ruleset repository is implemented as a DB2 database, the `db2jcc4.jar` JDBC driver available with the DB2 product installation might be selected to access the repository.

The type of the ruleset repository and the connect information for the repository is stored in the `ra.xml` JRules configuration file. The directory with this file needs to be listed in the class path for the connector. The JRules installation comes with the example `ra.xml` file in the *JRULES_HOME*/executionserver/bin directory, which can be customized for access to a particular ruleset repository. For more information about the `ra.xml` configuration file, refer to the ILOG JRules product documentation.

J2EE RES XU mode

The connection to the ruleset repository is configured when the Rule Execution Server is deployed to the J2EE server.

During the job execution, the connector acts as remote EJB3 client and establishes connection to the EJB3 session component that must be deployed to the J2EE server on which the Rule Execution Server is deployed. This engine mode requires that the ILOG JRules Java EE add-ons are installed and configured correctly in the JRules environment. The following JRules JAR files must be present in the class

path for the job execution, where JRULES_HOME represents the ILOG JRules product home directory on the IBM InfoSphere Information Server engine host.

JRULES_HOME/executionserver/lib/jrules-engine.jar

JRULES_HOME/executionserver/lib/jrules-res-session-java.jar

JRULES_HOME/executionserver/lib/jrules-res-execution.jar

JRULES_HOME/executionserver/lib/sam.jar

JRULES_HOME/executionserver/lib/log4j-1.2.8.jar

JRULES_HOME/executionserver/lib/j2ee_connector-1_5-fr.jar

In addition to the JRules JAR files, you must have the J2EE client JAR files necessary to establish connection to the EJB3 component on the J2EE server.

When the JRules engine is running on WebSphere Application Server v7 and the connector connects to it through the WebSphere Application Client v7 installed on the engine tier machine, you must add the following JAR files to the class path:

- *WAS_HOME*/runtimes/com.ibm.ws.ejb.thinclient_7.0.0.jar
- *WAS_HOME*/runtimes/com.ibm.ws.webservices.thinclient_7.0.0.jar

Where *WAS_HOME* is the WebSphere Application Client 7 home directory on the IBM InfoSphere Information Server engine tier machine.

Note: You must select the **Stand-alone Thin Clients and Resource Adapters** option during the WebSphere Application Client installation in order for these two JAR files to be included in the installation.

The EJB3 stub JAR files must be created and listed in the class path. This stub JAR file is generated by running the `createEJBStubs` script located in the `WAS_HOME/bin` directory, where `WAS_HOME` is the home directory of the WebSphere Application Server v7 installation on which the Rule Execution Server is deployed. The file extension for this script differs between operating systems. For example, on Windows, it is `.bat` and on Linux it is `.sh`. The script must be executed once against the `jrules-res-session-ejb3-WAS7.jar` file located under *JRULES_HOME*/executionserver/applicationservers/WebSphere7 directory. The option `-newfile` is to specify the location of the generated stub file. For example:

```
WAS_HOME/bin/createEJBStubs.sh JRULES_HOME/executionserver/
applicationservers/WebSphere7/jrules-res-session-ejb3-WAS7.jar -newfile
JRULES_HOME/executionserver/lib/jrules-res-session-ejb3-WAS7_stub.jar
```

The generated EJB3 stub file must be copied to the IBM InfoSphere Information Server engine host and included in the class path for the job execution.

When the ruleset for which the connector is configured is based on Java XOM, then the XOM classes of the ruleset parameters must also be included in the class path. This applies to the job runtime as well as the configuration wizard sessions. In this engine mode, the Java XOM classes must implement the `java.io.Serializable` interface. Otherwise, the jobs fail with CORBA marshalling exceptions as the ruleset parameter values are not transported properly over the EJB connection.

The connector also requires access to the `jndi.properties` configuration file. This file contains the properties needed to create the initial context (`java.naming.InitialContext` object) to establish connection to the EJB3 rule session component on the J2EE server. An example `jndi.properties` file is

provided with JRules installation under the *JRULES_HOME*/executionserver/samples/j2eerulesession/websphere7 directory. You can customize it and add its containing directory to the class path for the connector.

You must also point the connector to sas.client.props, ssl.client.props, and soap.client.props files which contain additional information for establishing connection to the WebSphere Application Server from the WebSphere Application Client installation used by the connector. The WebSphere Application Client installation has sample files that can be customized and configured to connect to a particular WebSphere Application Server instance. These sample files are stored in the *WAS_HOME*/properties directory.

Specify the following options in the **JVM settings** property under the **Java properties** section in the Stage properties:

```
-Dcom.ibm.CORBA.ConfigURL=file:WAS_HOME/properties/sas.client.props
-Dcom.ibm.SSL.ConfigURL=file:WAS_HOME/properties/ssl.client.props
-Dcom.ibm.SOAP.ConfigURL=file:WAS_HOME/properties/soap.client.props
-Djava.util.logging.manager=com.ibm.ws.bootstrap.WsLogManager
-Djava.util.logging.configureByServer=true
```

Where, *WAS_HOME* is the WebSphere Application Client 7 home directory on the IBM InfoSphere Information Server engine tier machine.

Note: The configuration wizard never accesses the JRules Rule Execution Server deployed to WebSphere Application Server. If the J2EE RES XU engine mode is selected when the configuration wizard is launched, or if this engine mode is selected in the configuration wizard session, the wizard accesses the JRules engine as the locally managed JRules Rule Execution Server. The requirements in this case are therefore the same as for J2SE RES XU engine mode.

Generating Java code through the configuration wizard

When the configuration wizard is used to generate Java code based on the columns defined on the stage links, the connector does not require access to any resources other than what is automatically provided by IBM InfoSphere Information Server and the Java Runtime Environment (JRE) in which the connector runs.

Designing a InfoSphere DataStage job with ILOG JRules connector

You must create an IBM InfoSphere DataStage job and configure it to invoke specific JRules ruleset.

Procedure

1. Create a job.
2. Configure the ILOG JRules connector stage.
3. Configure the input and output link properties.
4. (Optional)Configure the reject link property.
5. (Optional) Configure buffer links.
6. Compile the job.

Creating a job

You must create an IBM InfoSphere DataStage parallel job with the ILOG JRules stage to invoke specific JRules ruleset.

Procedure

1. From the Designer client, select **File** > **New**.
2. Select the **Parallel Job** icon, and click **OK**.
3. In the Parallel Job canvas, define stages for accessing your source and target data such as a file or database data.
4. In the Designer client palette area, click **Real Time**.
5. Select the **ILOG JRules Connector** stage icon and drag the stage to your open job. Position the stage in between the source and target stages.
6. Link the different stages.
7. (Optional) Rename the links and stages.
8. Select **File** > **Save** to save the job.

Configuring the stage to invoke the ruleset

When you create an ILOG JRules stage job, you must configure the stage properties.

Procedure

1. On the parallel canvas, double-click the ILOG JRules Connector stage icon.
2. On the **Properties** tab, specify a value for the Engine mode field.
3. Specify a value for the **Ruleset location** field.
4. To enable batch processing, select **Yes** as the value for the **Enable batch processing** field.
5. To enable key processing, select **Yes** as the value for the **Enable key processing** field.
6. Specify a value for the **XOM type** field.
7. To propagate unmatched fields from input links to output links, select **Yes** as the value for the **Propagate unmatched fields** option.
8. Specify a value for **Field propagation strategy**, depending on how you want the connector to associate the input and output links when propagating the unmatched input link fields.
9. In the **Java properties** section, set the **Classpath** field value to include directories and archives with Java class definitions and configuration files required by the stage.
10. (Optional) Specify a value for the **Heap size** field.
11. (Optional) Specify a value for the **JVM options** field.
12. Click **OK** to save.

Configuring input and output link properties

The ILOG JRules stage that invokes the selected JRules ruleset contains input and output links. The ILOG JRules stage supports multiple input and output links. In the key mode, input records must be sorted and correlated by a common key. There is a master input link and one or more secondary input links.

Procedure

1. Open the ILOG JRules connector stage.
2. Open the input link **Properties** tab.
3. In the **Ruleset parameter name** field, specify the name of the ruleset parameter as defined in the ruleset signature.

4. In the **Ruleset parameter class** field, specify the Java XOM class for the ruleset parameter that is associated with the link. If the Java XOM class is an array, then the parameter name must end with []. If the Java XOM class is a Dynamic XOM, this field must be left empty.
5. If the **Propagate unmatched fields** option is enabled in the stage properties and the **Field propagation strategy** option is set to **Match using key values present in records**, in the **Propagation Key Column** field, specify the propagation key column to be used.
6. (Optional) To convert the input link as a buffer link, set the **Buffer link** field to **Yes**.
7. Click **OK** to save.

Connector stage links and ruleset parameters association

The links of a ILOG JRules stage must be mapped to ruleset parameter names. If the ruleset is based on Java XOM, the fully-qualified Java class of the ruleset parameter must also be specified for each link. If the parameter is of array type then the class name must end with [].

ILOG JRules connector reads records from an input link and converts them to Java objects of the class the link is mapped to. If the parameter is of type array, then all objects derived from the same input link are stored in an array. The entire array is set as the parameter value.

If the parameter is not of an array type, then individual input records are converted to Java objects and set as the parameter values. If a parameter associated with the secondary link is not an array and multiple rows from the secondary link join a single row of the primary link, then the rows from the secondary link are sent one object at a time to ILOG JRules. The same row from the master link is sent to ILOG JRules for each ruleset invocation.

To produce records on the output links, design the rules to add Java objects to the Java arrays or variables associated with the output ruleset parameters. If input and output links are mapped to different classes, output objects must be created by the business rules. However, if the same Java class is mapped to input and output links, you can pass the input parameter object to output parameters. In this scenario, output objects are not created explicitly by the rules but are passed from input.

Buffer links

If an input link is marked as a buffer link, the connector reads the records from this link only once and reuses the records for every ruleset execution for the duration of the job. You can configure an input link in the ILOG JRules connector job as a buffer link

When buffer link is configured, the records captured in the buffer link are used to generate fixed ruleset parameter values to be used across multiple ruleset executions.

If the records arrive to the stage in transaction waves, the connector reads the records from the buffer links separately in each of the waves and reuses them across ruleset executions in that same wave.

If there are no records on the buffer link, action is taken depending on the type of the ruleset parameter associated with the buffer link. If the ruleset parameter is

based on an array Java type, the connector creates an empty array and uses it for the ruleset parameter value across ruleset executions. Otherwise, the connector logs an error and the job fails.

When the ruleset parameter associated with the buffer link is not based on an array Java type, or, if the stage is configured for a ruleset based on Dynamic XOM, the connector expects to find a single record on the buffer link and to use it to produce that ruleset parameter value. If multiple records are available on the link, the connector sends the records that follow the first record to a reject link, if defined for that input link and the **Leftover record error** option is enabled. Otherwise, the connector logs an error message and the job fails.

If a secondary link is configured as a buffer link, regardless of the mode in which the connector runs and regardless of the batch size and key columns specified for the stage, the connector reads all available records on that buffer link.

If the master link is configured as a buffer link, the connector first reads all available records on the master link. Then, it reads the records from the remaining secondary links which are not configured as buffer links as follows:

- If the connector is running in batch mode and the key mode is disabled, the connector reads all available records on the secondary link, but not more than *n* records, where *n* is the specified batch size (0 for unlimited). After the ruleset is executed, the connector continues to read records on this link for the next ruleset execution.
- If the connector is running in key mode, then the batch size value is ignored and the connector reads all the records on the secondary link and expects all of them to have key column values that match the key column values in one of the records retrieved on the master buffer link. A single ruleset execution is performed in the job, or in the wave if the records are arriving in transaction waves. The records whose key column values do not match the key column values of any of the master records are rejected if the reject link is defined for that secondary link and the **Key mismatch error** option is selected for that reject link. Otherwise the connector logs an error and the job fails.

Null value handling

ILOG JRules connector has a mechanism to handle null values in the fields of the records on the input and output links and the Java null values in the ruleset parameter values.

Input links

When a field of an input record used to produce the value of the IN or IN_OUT ruleset parameter is a null value, the null value is mapped to the Java null value in the ruleset parameter if that mapping is possible. If the mapping is not possible, the job fails with an error if a reject link is not configured. If a reject link is configured for the input link and the **Null input error** option is enabled, the record is rejected.

It is not possible to map the null input value to the Java null value in the ruleset parameter in the following scenarios:

- The ruleset is based on Dynamic XOM. In this case, the field that represents the XML document value of the ruleset parameter cannot be set to null.
- The field maps to the ruleset parameter of a primitive Java type such as byte, short, int, long, float, double, boolean or char. The null value cannot be specified for a primitive Java type.

- The field maps to an argument of a method in the Java class of the ruleset parameters, and that argument is of a primitive Java type. Again the null value cannot be specified for a primitive Java type.

Output links

When a Java null value is encountered while inspecting the value of the OUT or IN_OUT ruleset parameter associated with the output link, then the Java null value is mapped to the null value in the corresponding output record field, if the mapping is possible. If the mapping is not possible, the job fails with an error.

It is not possible to map the Java null value to the null value in the corresponding output record field in the following scenarios:

- The ruleset parameter value is null and it maps to a single column on the output link which is not nullable. A column is not nullable if it has the **Nullable** attribute set to **No**.
- The ruleset parameter value is null and it maps to multiple columns on the output link, of which some are nullable and some are not. In this case, the connector is unable to invoke methods on the output parameter to initialize field values based on the method return values.
- The ruleset parameter value is not null but a method invoked on the Java object of the output parameter returns null, and the column to which the return value is mapped is not nullable.

Configuring the reject link

You can configure reject links on the ILOG JRules stage to accept the input records that the connector found to be in error. When the connector sends the records in error to the reject links defined for the stage, the job continues to run for the remaining input records and you can verify the rejected records to determine the cause for error. If a reject link is not configured for an input link on which the connector detected records in error, the connector logs an error and the job fails.

Procedure

1. Define an output link for the ILOG JRules stage
2. Right-click the output link, then select the **Convert to reject** check box to convert the output link as a reject link.
3. Open the stage editor, select the **Reject** tab.
4. Select the relevant options under the **Reject rows based on selected** list to set the reject condition.
5. Select **ERRORCODE** or **ERRORTEXT**, or both, in the **Add to reject row** section to specify that the error code and error text information must be included in the rejected records to identify the reason for which the records were rejected.
6. In the **Reject From Link** field, select the input link.
7. In the **Abort when** field, specify the limit value for the condition.
8. In the **Abort after** field, specify the upper limit for reject links.
9. Click **OK** to save.

Compiling the job

When you finish designing a job, compile it.

Procedure

1. In the Designer client, open the job that you want to compile.
2. On the toolbar, click **Compile**.
3. If any error messages are displayed, then edit the job to resolve the errors. Click **Compile** after resolving the errors to recompile the job. After compiling the job successfully, you can run it either from IBM InfoSphere DataStage and QualityStage® Designer, or from IBM InfoSphere DataStage and QualityStage Director.

Using the configuration wizard

ILOG JRules connector includes a configuration wizard that can be used to simplify the configuration of properties and link schemas in the stage to invoke the JRules ruleset when the job runs. The configuration wizard can also be used to automatically generate Java code based on the link schemas and property values defined on the stage.

Configuring the stage by using the wizard

ILOG JRules connector includes a wizard that you can use to configure properties and link schemas in the stage to invoke the selected JRules ruleset when the job runs.

About this task

When you configure the stage to invoke a specific JRules ruleset, the number of input links must match the number of IN and IN_OUT ruleset parameters, and the number of output links must match the number of OUT and IN_OUT parameters. Reject links on the stage are excluded from this formula and are ignored by the configuration wizard. If you know the number and type of ruleset parameters in advance, you can define the correct number of links of correct types on the stage before invoking the wizard. If the information is not available, you can still launch the wizard and use it to locate the ruleset of interest and find out the number and type of parameters defined for it. However, if there are missing links, then you have to cancel the wizard and manually add the required links in order for the wizard to be able to complete the stage configuration because the wizard cannot automatically add any links to the stage.

Procedure

1. Double-click the ILOG JRules stage.
2. Click **Configure**.
3. On the **Wizard action and log file settings** page, select **Import ruleset parameters**.
4. (Optional) Specify the path of the log file in the **Log file path** field to store the debug information, then click **Next**.
5. On the **Engine mode and ruleset filter** page, specify the engine mode and ruleset filter expression, then click **Next**.
6. Configure the subsequent screen depending on engine mode selected.
   - If you selected **Core engine** as engine mode, the **Ruleset Archive selection page** is displayed. Select the **Include inherited methods** check box to specify whether to process methods that the ruleset parameter Java classes inherit from their ancestor classes. Specify the ruleset archive file for which to configure the stage.

- If you selected either **J2SE RES XU** or **J2EE RES XU** as engine mode, the **Deployed ruleset selection** page is displayed. Select the **Include inherited methods** check box to specify whether to process methods that the ruleset parameter Java classes inherit from their ancestor classes. Specify the deployed ruleset for which to configure the stage.

7. On the **Ruleset Parameters selection** page, configure the import mode and the ruleset parameters and the corresponding Java class methods for which to configure schema definitions for the links defined on the stage.
8. On the **Schema mapping for the links and ruleset parameters** page, define the associations among the ruleset parameters and the links of the stage.
9. On the **Import ruleset parameters preview** page, review the report and the summary of expected results and click **Finish**.

Generating Java code for the stage by using the wizard

You can use the wizard to generate Java code based on the column definitions of the links defined on the stage.

Procedure

1. Double-click the ILOG JRules connector.
2. Click **Configure**.
3. On the **Wizard action and log file settings** page, select the **Generate Java code** value in the **Wizard action** field.
4. On the **Java code settings** page, in the **Source code directory path** field, specify the fully-qualified directory path on the engine host to which to write the generated `.java` files.
5. In the **Output format** field, specify the format of the Java type for which to generate the code.
6. In the **Member variable prefix** field, specify the prefix to use for the generated private member variable names.
7. Select the **Replace existing files** check box to specify whether to overwrite the `.java` files that have the same name and location.
8. Select the **Exclude individual methods** check box to specify whether to allow manual exclusion of individual methods from the list of reported methods so that code is not generated for those methods.
9. In the **Source code mapping for the Java types** page, inspect the Java methods for which the code will be generated. If, in the previous step the **Exclude individual methods** option was selected, select the methods for which you do not want the code to be generated. Click **Next**.
10. In the **Generate Java code preview** page, review the actions that the wizard will perform, then click **Finish**.

Results

The files are written to the subdirectories of the specified source code directory determined by the package specifications in the Java type names. If the package is not specified for some of the types, the `.java` source code file for that type is generated in the specified source code directory. For example if the source code directory is specified as `/projects/javaxom` and the Java type name is `com.example.import.Test`, then `Test.java` file is created in the `/projects/javaxom/com/example/import` directory. If the directory exists, the wizard reuses it. If a problem is encountered during the creation of the directory or while writing the `.java` file to the directory, an error message is displayed.

Wizard configuration file

The wizard configuration file provides the option to specify the default values displayed by the configuration wizard when it is launched from the connector stage.

Purpose

In some cases, the default values for the configuration wizard settings might not be convenient with respect to the current environment. For example, if the wizard is primarily launched to generate Java code for the column definitions on the links, then the **Generate Java code** is a better default option than **Import ruleset parameters** in the **Wizard action and log file settings** page.

In some cases the default value may be specific to the current system environment. For example if the wizard is primarily used to configure the stage based on the ruleset archive files which in turn are always deployed to the same directory on the InfoSphere Information Server engine host, the path to that directory may be a good candidate to use as the default value for the **Ruleset filter expression** setting.

The default values for the wizard settings can be specified as properties in the wizard configuration file `ccjrules.wizard.properties`. This file is not created by default when the connector stage type is installed. You can create and copy the wizard configuration file to the *IS_HOME*`/ASBNode/lib/java` directory on the InfoSphere Information Server engine host, where, *IS_HOME* is the InfoSphere Information Server home directory. Alternatively, you can include the directory in the **Classpath** property value on the stage from which the wizard is launched.

An example content for the `ccjrules.wizard.properties` file is show in the Sample section. The sample content include all the wizard settings for which the default value can be specified. The lines that assign default values to these settings are all disabled by default. To enable a particular setting, the # character at the beginning of the line needs to be removed.

For example, to specify that the directory `C:/temp/javadoc` is used as the default directory in which the wizard should generate `.java` source code files, the initial character # must be removed from the line:

```
# source_code_directory = C:/temp/javacode
```

Each time the content of the wizard configuration is changed, you must close and launch the wizard again in order to use the new values.

Sample

You can copy the code sample given below to create your own wizard configuration file:

```
#-------------------------------------------------------------------------------
# Default wizard action:
# import_ruleset - Import ruleset parameters
# generate_code - Generate Java code
#-------------------------------------------------------------------------------
# wizard_action = import_ruleset
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Log file location. Use forward slash as file separator on all platforms,
# including Windows.
```

```
#-------------------------------------------------------------------------------
# log_file_path = C:/temp/ccjrules.log
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Overwrite log file flag. The allowed values are:
# 0 - Do not overwrite. Append to the log if it exists, otherwise create it.
# 1 - Overwrite the file if it exists, otherwise create it
#-------------------------------------------------------------------------------
# overwrite_log_file = 0
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Default engine mode. If the value is specified here it takes precedence
# over the Engine mode property value in the stage from which the wizard was
# invoked. The allowed values are:
# core_engine - Core engine
# j2se_res_xu - J2SE RES XU
# j2ee_res_xu - J2EE RES XU
#-------------------------------------------------------------------------------
# engine_mode = core_engine
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Default filter expression for locating rulesets
# If specified, the value is used unless a value was specified in the
# Ruleset location property in the stage from which the wizard was invoked. In
# that case the Ruleset location value is used as the default filter expression
# in the wizard. Use forward slash as the file separator on all platforms,
# including Windows.
#-------------------------------------------------------------------------------
# filter_expression = C:/temp
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Default preference for displaying all ruleapp and ruleset versions. The
# allowed values are:
# 0 - Show all ruleapp and ruleset versions
# 1 - Show only the most recent ruleapp and ruleset versions
#-------------------------------------------------------------------------------
# show_all_versions = 0
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Default preference for displaying inherited methods for ruleset parameter
# classes. The allowed values are:
# 0 - Do not include inherited methods
# 1 - Include inherited methods
#-------------------------------------------------------------------------------
# include_inherited_methods = 0
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Default import mode for preserving existing columns on the link when
# importing ruleset parameters. The allowed values are:
# merge - Utilize the existing columns for the mapping
# replace - Replace all the existing columns
#-------------------------------------------------------------------------------
# import_mode = merge
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Directory in which to generate .java files. Use forward slash as file
# separator on all platforms, including Windows.
#-------------------------------------------------------------------------------
# source_code_directory = C:/temp/javacode
#-------------------------------------------------------------------------------
```

```
#-------------------------------------------------------------------------------
# Output format for the generated source code. The allowed value are:
# class - Java class
# inteface - Java interface
#-------------------------------------------------------------------------------
# output_format = class
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Prefix for member variables in the generated code
#-------------------------------------------------------------------------------
# member_variable_prefix = m_
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Default preference for replacing the existing .java files when generating
# the .java code.
# 0 - Do not replace the existing files
# 1 - Replace the existing files
#-------------------------------------------------------------------------------
# replace_existing_files = 0
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Default preference for excluding individual methods when generating the
# .java code.
# 0 - Do not enable the exclusion of individual methods
# 1 - Enable the checkboxes to use to exclude individual methods
#-------------------------------------------------------------------------------
# exclude_individual_methods = 0
#-------------------------------------------------------------------------------
```

Basic Java types and methods

Basic Java type in the context of the ILOG JRules connector is a Java type that can be mapped directly to a link column.

Basic Java types

Basic Java types are:

- Primitive types: int, short, long, double, float, boolean, byte, char
- Wrapper classes for primitive types: java.lang.Integer, java.lang.Short, java.lang.Long, java.lang.Double, java.lang.Float, java.lang.Boolean, java.lang.Byte, java.lang.Character
- String type: java.lang.String
- Date/time types: java.util.Date, java.util.Calendar, java.sql.Date, java.sql.Time, java.sql.Timestamp
- Numeric types: java.math.BigInteger, java.math.BigDecimal

When a ruleset parameter is an XML parameter or a Java parameter of basic Java type, the parameter is mapped directly to a link column as follows:

- If the parameter is mapped to a column on an input link, then that parameter must be an IN or IN_OUT parameter. The column represents the value to be passed to the parameter when the ruleset is invoked.
- If the parameter is mapped to a column on an output link, then that parameter must be an OUT or IN_OUT parameter. The column represents the value returned by the ruleset for the parameter, when the ruleset is invoked.

When the ruleset parameter does not map directly to a link column, the ruleset constructs that map to the link columns are the return values and arguments of basic methods of the Java class associated with the ruleset parameter.

Basic Java methods

A basic Java method meets one of the following conditions:

- It is a constructor method or a method that does not return a value (void method), and all its arguments are of basic Java types. The method arguments are mapped to the input link column associated with the ruleset parameter in whose class the method is defined. The ruleset parameter must be an IN or IN_OUT parameter. The columns values are passed to the method arguments when invoking the method on the respective ruleset parameter object before invoking the ruleset and passing the parameter to it.
- It is a method that does not contain any arguments and its return value is a basic Java type. The method return value maps to a column on the output link associated with the ruleset parameter in whose class type that method is defined. The ruleset parameter must be an OUT or IN_OUT parameter. The column represents the return value to be retrieved from the method after the ruleset containing the parameter is invoked and the method is invoked on that ruleset parameter object.

The configuration wizard handles ruleset parameters associated with Java array types as if they were of the corresponding non-array types. When the job is run, the stage typically collects multiple records on the link to prepare the ruleset parameter value. The data records on the link associated with that ruleset parameter correspond to Java objects that are stored in the array which as a whole corresponds to the ruleset parameter value. However, if a ruleset parameter is of char[] and byte[] Java array types, then it corresponds to a single field value from a single record on the link, as follows:

- If the column is of character data type such as VarChar or NVarChar, then the characters of the text field value are mapped to the char elements of the char[] array for the ruleset parameter.
- If the column is of a binary data type such as VarBinary then the bytes of the corresponding binary field value map to the byte elements of the byte[] array.

DataStage type to Java type mapping

The wizard maps InfoSphere DataStage column type definitions in the link to the Java types used for the private member variables, method return values, and method arguments when it generates Java code.

At runtime, the connector maps, as necessary, the InfoSphere DataStage column type definitions present on the input links to the Java types, in order to initialize values for the IN and IN_OUT parameters associated with those links. The following table describes how the InfoSphere DataStage type definition is mapped to Java type:

Table 1. InfoSphere DataStage type to Java type mapping

DataStage SQL Type	Length	Scale	Extended	Java Type
TinyInt	(n/a)	(n/a)	Empty	byte, java.lang.Byte
TinyInt	(n/a)	(n/a)	Unsigned	short, java.lang.Short
SmallInt	(n/a)	(n/a)	Empty	short, java.lang.Short

Table 1. InfoSphere DataStage type to Java type mapping (continued)

DataStage SQL Type	Length	Scale	Extended	Java Type
SmallInt	(n/a)	(n/a)	Unsigned	int, java.lang.Integer
Integer	(n/a)	(n/a)	Empty	int, java.lang.Integer
Integer	(n/a)	(n/a)	Unsigned	long, java.lang.Long
BigInt	(n/a)	(n/a)	Empty	long, java.lang.Long
BigInt	(n/a)	(n/a)	Unsigned	java.math.BigInteger
Bit	(n/a)	(n/a)	(n/a)	boolean, java.lang.Boolean
Float	(n/a)	(n/a)	(n/a)	float, java.lang.Float
Double	(n/a)	(n/a)	(n/a)	double, java.lang.Double
Decimal	Any	Any	(n/a)	java.math.BigDecimal
Char, NChar	1	(n/a)	Any	char, java.lang.Character
Char, NChar	n > 1	(n/a)	Any	char[]
Char, NChar	Empty	(n/a)	Any	char[]
VarChar, NVarChar, LongVarChar, LongNVarChar	Any	(n/a)	Any	java.lang.String
Binary	1	(n/a)	(n/a)	byte, java.lang.Byte
Binary	n > 1	(n/a)	(n/a)	byte[]
Binary	Empty	(n/a)	(n/a)	byte[]
VarBinary	Any	(n/a)	(n/a)	byte[]
Time	(n/a)	(n/a)	Empty	java.sql.Time
Time	(n/a)	(n/a)	Microseconds	java.sql.Timestamp
Date	(n/a)	(n/a)	(n/a)	java.sql.Date
Timestamp	(n/a)	(n/a)	Empty	java.util.Date
Timestamp	(n/a)	(n/a)	Microseconds	java.sql.Timestamp

Some of the InfoSphere DataStage type definitions can be mapped to more than one Java type. In such scenarios, the value specified for the **Nullable** attribute determines which of the two Java types is mapped. If the **Nullable** value is set to **No**, the primitive Java type is selected. If the **Nullable** value is set to **Yes** or **Unknown**, the object wrapper type is selected. For example, if a link column of InfoSphere DataStage TinyInt type with **Signed** attribute set to **Yes** maps to the Java primitive type `byte`, if the **Nullable** attribute is set to **No** and maps to Java wrapper type `java.lang.Byte` if its **Nullable** attribute is set to **Yes**.

InfoSphere DataStage character-based data types Char, NChar, VarChar, LongVarChar, NVarChar, and LongNVarChar can also be mapped to `boolean` and `java.lang.Boolean` Java types. The text value "true" (not case-sensitive) maps to boolean value *true.* The remaining text values map to boolean value *false.*

Java type to DataStage type mapping

The configuration wizard maps Java types used by the ruleset parameters to the corresponding column definitions on the links of the stage.

At runtime, the connector maps, as necessary, the Java types in OUT and IN_OUT ruleset parameters associated with output links to InfoSphere DataStage type definitions for the columns on those links.

Table 2. Java type to InfoSphere DataStage type mapping

Java type	DataStage SQL Type	Length	Scale	Extended
short, java.lang.Short	SmallInt	(n/a)	(n/a)	Empty
int, java.lang.Integer	Integer	(n/a)	(n/a)	Empty
long, java.lang.Long	BigInt	(n/a)	(n/a)	Empty
float, java.lang.Float	Float	(n/a)	(n/a)	(n/a)
double, java.lang.Double	Double	(n/a)	(n/a)	(n/a)
boolean, java.lang.Boolean	Bit	(n/a)	(n/a)	(n/a)
byte, java.lang.Byte	TinyInt	(n/a)	(n/a)	Empty
byte[], java.lang.Byte[]	VarBinary	Empty	(n/a)	(n/a)
char, java.lang.Character	Char	1	(n/a)	Unicode
char[], java.lang.Character[]	VarChar	Empty	(n/a)	Unicode
java.lang.String	VarChar	Empty	(n/a)	Unicode
java.util.Date, java.util.Calendar	Timestamp	(n/a)	(n/a)	Microseconds
java.sql.Date	Date	(n/a)	(n/a)	(n/a)
java.sql.Time	Time	(n/a)	(n/a)	Microseconds
java.sql.Timestamp	Timestamp	(n/a)	(n/a)	Microseconds
java.math.BigInteger	BigInt	(n/a)	(n/a)	Empty
java.math.BigDecimal	Decimal	255	127	(n/a)

The link columns that correspond to int, short, long, double, float, boolean, byte, and char Java types are configured with the **Nullable** attribute set to **No**. The columns for the remaining Java types, including char[] and byte[] types, are configured with the **Nullable** attribute set to **Yes**.

The `boolean` and `java.lang.Boolean` Java types can also be mapped to character-based InfoSphere DataStage data types Char, NChar, VarChar, LongVarChar, NVarChar, and LongNVarChar. The boolean value true is mapped to text value *true*, and the boolean value false is mapped to text value *false*.

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:

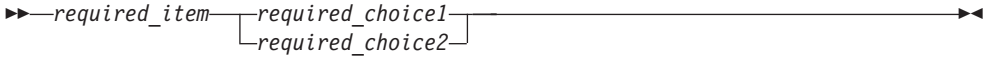
- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
  - The >>--- symbol indicates the beginning of a syntax diagram.
  - The ---> symbol indicates that the syntax diagram is continued on the next line.
  - The >--- symbol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

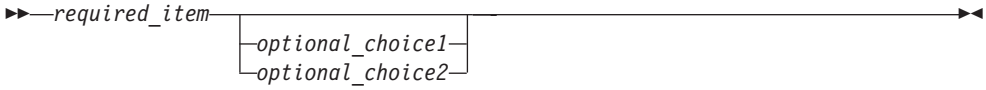
  If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.

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

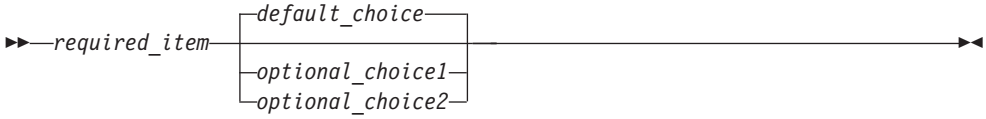
  If you must choose one of the items, one item of the stack appears on the main path.



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

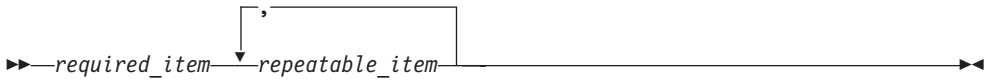


  If one of the items is the default, it appears above the main path, and the remaining choices are shown below.



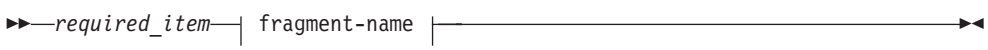
- An arrow returning to the left, above the main line, indicates an item that can be repeated.

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 3. 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 4. 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 • Persistent	User name	• Session management • Authentication	Cannot be disabled
Any (part of InfoSphere Information Server installation)	InfoSphere Metadata Asset Manager	• Session • Persistent	No personally identifiable information	• Session management • Authentication • Enhanced user usability • Single sign-on configuration	Cannot be disabled

Table 4. 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 • Persistent	• User name • Digital signature • Session ID	• Session management • Authentication • Single sign-on configuration	Cannot be disabled
InfoSphere DataStage	XML stage	Session	Internal identifiers	• Session management • Authentication	Cannot be disabled
InfoSphere DataStage	IBM InfoSphere DataStage and QualityStage Operations Console	Session	No personally identifiable information	• Session management • Authentication	Cannot be disabled
InfoSphere Data Click	InfoSphere Information Server web console	• Session • Persistent	User name	• Session management • Authentication	Cannot be disabled
InfoSphere Data Quality Console		Session	No personally identifiable information	• Session management • Authentication • Single sign-on configuration	Cannot be disabled
InfoSphere QualityStage Standardization Rules Designer	InfoSphere Information Server web console	• Session • Persistent	User name	• Session management • Authentication	Cannot be disabled
InfoSphere Information Governance Catalog		• Session • Persistent	• User name • Internal identifiers • State of the tree	• Session management • Authentication • 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

SC19-4338-00