Designing Intelligent Event Processor (IEP) Projects

Part No: 821–0139
June 2009

100217@23474

Contents

◆ ◆ ◆ CHAPTER 1

Designing Intelligent Event Processor (IEP) Projects

The topics listed here provide information about how to use the Intelligent Event Processor (IEP).

If you have any questions or problems, see the Java CAPS web site at `http://goldstar.stc.com/support`.

What You Need to Know

What You Need to Do

Intelligent Event Processor Overview

The Intelligent Event Processor (IEP) enables you to perform complex event processing (CEP) and event stream processing (ESP) from within an enterprise service bus.

Complex Event Processing and Event Stream Processing

In the most general sense, the term *event* refers to anything that happens in a system. For example:

- A password change
- A stock purchase
- A transfer of funds

You use IEP to process computerized representations of these events.

These events are generated and sent out by applications. The applications can be located within the enterprise service bus, or they can come from an external system that is connected to the enterprise service bus.

The term *event stream* refers to a continuous set of events. For example, an event stream could contain the password changes made by the users of a web-based application.

Processing an event stream can involve many types of activities. For example:

- You can examine a bounded portion of an event stream, such as all of the events that occurred in the last two minutes. This bounded portion is called a *window*.
- You can apply a function to a set of events. For example, you can determine the average price of a stock during the last three hours, with the calculation updated every five minutes.
- You can change the order of the events.

When you combine multiple events to create a higher level event, the result is called a *complex event*.

An architectural style in which software modules operate in response to the arrival of events is called *event driven architecture*.

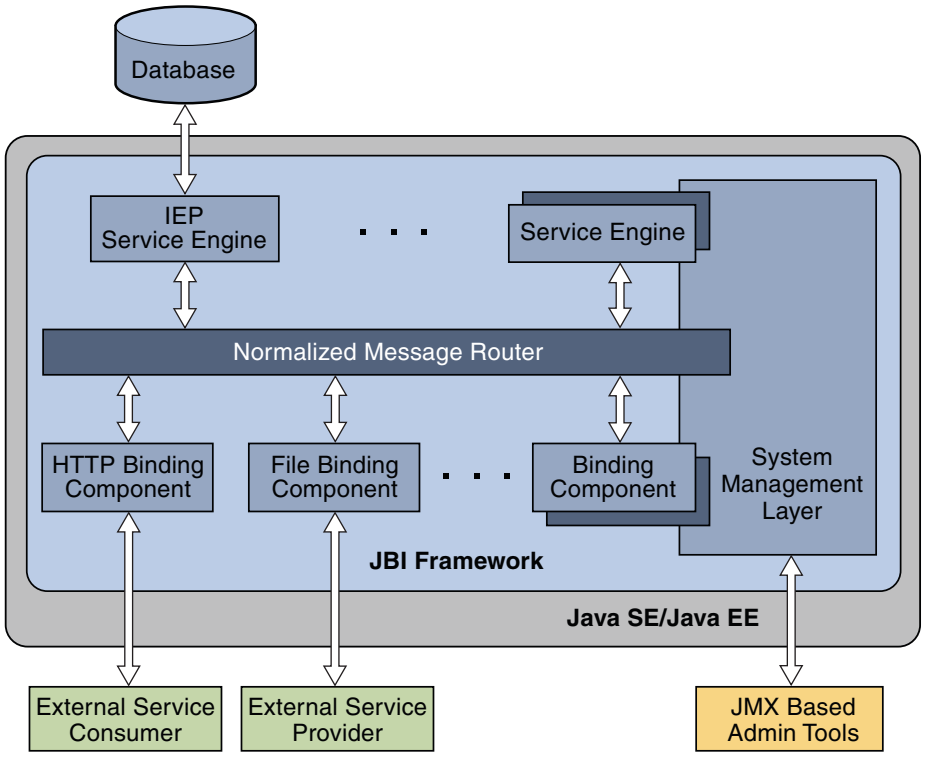
Typical IEP Scenarios

The following table describes typical IEP scenarios.

Scenario	Example
Financial trade auditing and compliance	Examine a stream of stock transactions and find any transactions that are suspicious. Check whether any traders involved in a suspicious transaction also appear in a database table that contains the names of persons of interest.
Network monitoring and traffic engineering	Receive an undifferentiated stream of alerts from various hardware devices, group them by the device, and sort them by the unique ID. For each device, detect any missing alerts and request them to be re-sent.
IT security event correlation	Examine the password changes that have been made to a web-based application. If the number of password changes in a given hour is more than twice the average, then generate a security alert.
Asset management and tracking using RFID	Examine the RFID signals that are regularly emitted by all of the products in a store. Determine whether a product is moving through the exit area without having been purchased. Determine whether a product's RFID emitter is no longer working.

IEP Architecture

The following diagram illustrates the IEP architecture.



Within the enterprise service bus, the IEP Service Engine can interact with any Java™ Business Integration (JBI) service engine or binding component that is also plugged into the bus. The Normalized Message Router takes care of message exchanges between the components.

By default, the IEP Service Engine receives input events from the HTTP Binding Component and sends output to the File Binding Component. The IEP Service Engine uses a database to maintain information about deployed event processors.

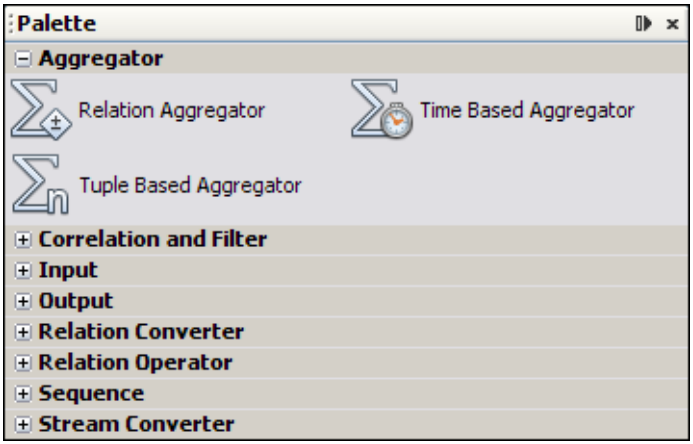
IEP Design-Time and Runtime Components

IEP consists of a design-time component and a runtime component.

- The design-time component is integrated within the NetBeans IDE.
- The runtime component is implemented as a JBI service engine.

In the NetBeans IDE, you create an IEP Module project and then add one or more event processors.

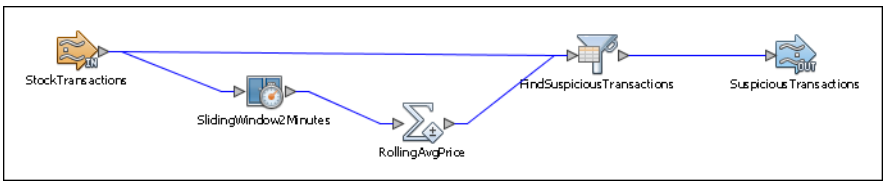
For each event processor, you drag operators from the palette onto the design canvas. In the palette, the operators are grouped into the following categories: Aggregator, Correlation and Filter, Input, Output, Relation Converter, Relation, Sequence, and Stream Converter.



An event processor must have at least one input operator and one output operator. You can add any number of operators between the input operator and the output operator.

On the design canvas, you connect the operators with each other and use property editors to configure the operators.

The following screen capture shows a set of operators in an event processor. The operator at the left is an input operator. The operator at the right is an output operator.



Some operators allow you to enter SQL-like statements. For these operators, knowledge of the SQL SELECT statement can be useful.

When you save an IEP Module project, IEP generates a Web Services Description Language (WSDL) document for each event processor. The WSDL documents contain the endpoints for the event processors.

You can run a set of predefined validation rules on an event processor at design time.

To deploy an IEP Module project, you must create and build a Composite Application project. These tasks create a service assembly. The service assembly is a collection of service units intended for the IEP Service Engine and any other required JBI component (such as the HTTP Binding Component and the File Binding Component).

When the project is deployed, the IEP Service Engine receives and processes the input events.

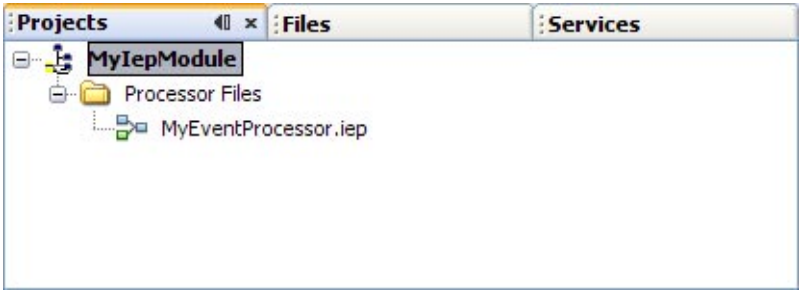
Basic Workflow

This topic describes the basic workflow of IEP.

Creating an Intelligent Event Processing Module Project

You create a new Intelligent Event Processing Module project in the NetBeans IDE. You then add one or more event processors to the project.

The following screen capture shows an IEP Module project that has one event processor.



▼ To Create an Intelligent Event Processing Module Project

1. **From the IDE's main menu, choose File → New Project.**
   The New Project wizard opens.
2. **In the Categories list, select the SOA node.**
3. **In the Projects list, select the Intelligent Event Processing Module node.**
4. **Click Next.**

5 **(Optional) In the Project Name field, change the default project name.**

6 **Click Finish.**

The new IEP Module project appears in the Projects window. You can now add one or more event processors to the project.

▼ To Add an Event Processor to the Project

1 **Right-click the Processor Files node and choose New → Intelligent Event Processor.**

2 **(Optional) In the File Name field, change the default file name.**

3 **Click Finish.**

The event processor is added. The IEP Editor opens in Design view. You can now define the event processing logic by adding and configuring IEP operators.

Adding and Configuring IEP Operators

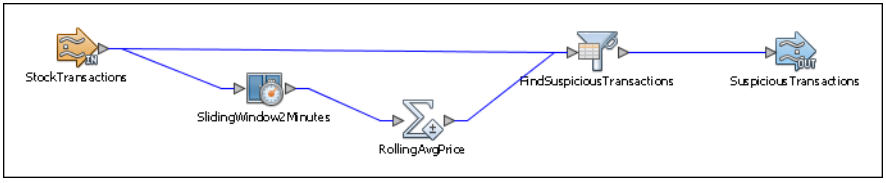
Once you add an event processor to an IEP Module project, you define the event processing logic by adding and configuring IEP operators.

An event processor must have at least one input operator.

An event processor must have at least one output operator.

You can add any number of operators between the input operator and the output operator.

The following screen capture shows a set of operators in an event processor. The operator at the left is an input operator. The operator at the right is an output operator.



For general information about the IEP operators, see "Introduction to IEP Operators" on page 17.

For specific information about each IEP operator, see the following topics:

- "Aggregator Operators" on page 20

- "Correlation and Filter Operators" on page 24
- "Input Operators" on page 28
- "Output Operators" on page 34
- "Relation Converter Operators" on page 39
- "Relation Operators" on page 42
- "Sequence Operators" on page 46
- "Stream Converter Operators" on page 49

▼ To Add IEP Operators to an Event Processor

1. **If the Palette is not visible in the NetBeans IDE, then choose Window → Palette from the IDE's main menu.**
2. **Drag the appropriate IEP operators from the Palette to the Design view.**

▼ To Configure IEP Operators in an Event Processor

1. **For each IEP operator that you added, do the following:**
   a. **Double-click the operator. Alternately, you can click the operator, go to the Properties window, and click the ellipsis.**
      The property editor opens.
   b. **Edit the appropriate properties. For some operators, you can add or modify attributes.**
   c. **(Optional) Click the Documentation tab and supply notes on this component.**
   d. **Click OK.**
2. **Connect the operators to each other as appropriate.**
   You send the output of one operator to the input of another operator by selecting the output port of the first operator and dragging it to the input port of the second operator.

Disabling the Generation of Bindings and Services

When you save an IEP Module project, IEP generates a Web Services Description Language (WSDL) document for each event processor. The WSDL documents contain the endpoints for the event processors.

By default, the WSDL documents include concrete information (that is, bindings and services). You might need to manually update the default settings for any binding and service. For example, the default file directory is a Windows path, which would not work correctly on a

UNIX system. If you manually update the WSDL document and then save the IEP Module project again, the changes that you made to the WSDL document are overwritten.

You can configure IEP to generate abstract WSDL documents instead. The bindings and services are not included. With this approach, you can define the bindings and services by using the Composite Application Service Assembly (CASA) Editor. These bindings and services are not affected by subsequent changes to the IEP Module project.

▼ To Disable the Generation of Bindings and Services

1. **Go to the location of the IEP Module project and open the `project.properties` file.**
2. **Change the value of the `always.generate.abstract.wsdl` flag to `true`.**
3. **Save the `project.properties` file.**

Validating Event Processors

You can run a set of predefined validation rules on an event processor at design time. The validation rules include:

- An event processor must have at least one input operator.
- An event processor must have at least one output operator.
- All of an operator's required properties must have values.
- If an attribute has a data type of VARCHAR, then the size must also be specified.

▼ To Validate Event Processors

1. **In the NetBeans IDE, open the event processor that you want to validate.**
2. **Click the Validate XML icon at the top of the Design view.**
   The results appear in the Output window.
3. **If you switch to the Source view and click a validation error in the Output window, then the source of the error is highlighted.**

Creating and Deploying the Composite Application Project

You deploy an IEP Module project as part of a Composite Application project.

The Composite Application Service Assembly (CASA) Editor provides a visual interface for editing the deployment configuration of a Composite Application project. You can perform such tasks as adding JBI module projects, adding and removing connections between endpoints, and adding concrete WSDL elements.

To Create a Composite Application Project

1. **From the IDE's main menu, choose File → New Project.**
   The New Project wizard opens.
2. **In the Categories list, select the SOA node.**
3. **In the Projects list, select the Composite Application node.**
4. **Click Next.**
5. **(Optional) In the Project Name field, change the default project name.**
6. **Click Finish.**
   The new Composite Application project appears in the Projects window. In addition, the CASA Editor appears.

To Add the IEP Module Project to the Composite Application Project

1. **In the Projects window, right-click the Composite Application project and choose Add JBI Module.**
   The Select Project dialog box appears.
2. **Select the IEP Module project.**
3. **Click Add Project JAR Files.**
   The IEP Module project is added to the JBI Modules area of the CASA Editor.

To Define the Binding Components and Connections

1. **If you generated an abstract WSDL document, then use the CASA Editor to define the binding components and connections.**
   For detailed instructions, see the CASA Editor topics in the NetBeans IDE help.
2. **If you generated a concrete WSDL document, then you can override the generated bindings by deleting the connections and defining the new binding components and connections.**

Caution – For both abstract and concrete WSDL documents, do not clone the WSDL document to customize its generated bindings and services. Updates to the generated WSDL documents will not be updated after the cloning. Over time, the cloned and edited WSDL document deployed for the bindings will become inconsistent with the WSDL document deployed for the IEP Service Engine.

▼ To Deploy the Composite Application Project

1. **Right-click the Composite Application project and select Build.**
2. **Right-click the Composite Application project and select Deploy.**
3. **Wait until the `BUILD SUCCESSFUL` message appears in the Output window.**

   You can now create test cases to ensure that the IEP Module project works as expected.

   If you make additional changes to the IEP Module project, then you must rebuild and redeploy the Composite Application project.

Introduction to IEP Operators

The IEP operators enable you to define the logic in an event processor.

Most of the IEP operators take a stream or a relation as input and produce a stream or a relation as output.

- "Understanding Schemas" on page 17
- "Understanding Streams" on page 18
- "Understanding Relations" on page 18
- "Supported Data Types" on page 19
- "IEP Operator Inputs and Outputs" on page 20

Understanding Schemas

A *schema* defines the types of information that a set of data contains. A schema consists of one or more attributes. Each attribute is identified by a name and has a data type that specifies the allowed values.

For example, a schema could consist of the following attributes:

- An attribute called `Symbol` that allows character values of between 1 and 10 characters
- An attribute called `Price` that allows floating-point values

Understanding Streams

A *stream* is a series of timestamped events that have the same schema.

Assume that a stream has the following characteristics:

- Each event represents a stock transaction.
- The schema consists of the stock symbol and the stock price.

The following table shows an example of the events in the stream during a brief interval of time.

Symbol	Price	Timestamp
ADBE	21.60	2008-12-15-T10:30:02:899-05.00
AMZN	50.12	2008-12-15-T10:32:44:674-05.00
ATT	23.88	2008-12-15-T10:35:17:198-05.00
ADBE	21.70	2008-12-15-T10:38:23:257-05.00

Understanding Relations

A *relation* is a collection of events that match a user-defined condition at a point in time.

You can define the condition in various ways. For example:

- All events that arrived in the last five seconds
- All events that arrived in the last five hours
- The most recent two events

Assume that you define the condition as all events that arrived in the last five seconds. For the example in "Understanding Streams" on page 18, the relation at time 2008-12-15-T10:35:00:000-05.00 would consist of the following events.

Symbol	Price	Timestamp
ADBE	21.60	2008-12-15-T10:30:02:899-05.00
AMZN	50.12	2008-12-15-T10:32:44:674-05.00

Let's move forward one second in time. The relation at time 2008-12-15-T10:36:00:000-05.00 would consist of the following events. Compared with the previous relation, one event has dropped out and one event has been added.

Symbol	Price	Timestamp
AMZN	50.12	2008-12-15-T10:32:44:674-05.00
ATT	23.88	2008-12-15-T10:35:17:198-05.00

The relation at time 2008-12-15-T10:37:00:000-05.00 would consist of the following events. This relation has the same events as the previous relation.

Symbol	Price	Timestamp
AMZN	50.12	2008-12-15-T10:32:44:674-05.00
ATT	23.88	2008-12-15-T10:35:17:198-05.00

The relation at time 2008-12-15-T10:38:00:000-05.00 would consist of the following events. Compared with the previous relation, one event has dropped out.

Symbol	Price	Timestamp
ATT	23.88	2008-12-15-T10:35:17:198-05.00

The relation at time 2008-12-15-T10:39:00:000-05.00 would consist of the following events. Compared with the previous relation, one event has been added.

Symbol	Price	Timestamp
ATT	23.88	2008-12-15-T10:35:17:198-05.00
ADBE	21.70	2008-12-15-T10:38:23:257-05.00

A relation can be empty. For the example in "Understanding Streams" on page 18, the relation at time 2008-12-15-T10:45:00:000-05.00 would not contain any events because none of the events arrived in the last five seconds.

Supported Data Types

In the property editor of an IEP operator, you can assign any of the following data types to an attribute:

- INTEGER
- BIGINT
- DOUBLE

- VARCHAR
- DATE
- TIMESTAMP

The Size and Scale columns are disabled for the DATE and TIMESTAMP data types.

You must ensure that input and output data types are appropriately matched.

IEP Operator Inputs and Outputs

You can categorize the IEP operators by their input and outputs.

- **Stream to Relation**. Operators that take a stream as input, and produce a relation as output with the same schema as the stream.
- **Relation to Stream**. Operators that take a relation as input, and produce a stream as output with the same schema as the relation.
- **Relation to Relation**. Operators that take one or more relations as input, and produce a relation as output.
- **Stream to Stream**. Operators that take a stream as input, and produce a stream as output. The input stream and the output stream can have different schemas.
- **Relation to Table**. Operators that take a relation as input, and produce a table as output.

For specific information about each IEP operator, see the following topics:

- "Aggregator Operators" on page 20
- "Correlation and Filter Operators" on page 24
- "Input Operators" on page 28
- "Output Operators" on page 34
- "Relation Converter Operators" on page 39
- "Relation Operators" on page 42
- "Sequence Operators" on page 46
- "Stream Converter Operators" on page 49

Aggregator Operators

Aggregator operators enable you to aggregate data and to perform additional operations on that data to obtain output.

The following table lists the input and output for each operator.

Operator	Input	Ouput
Relation Aggregator	Relation	Relation

Operator	Input	Ouput
Time Based Aggregator	Stream	Stream
Tuple Based Aggregator	Stream	Stream

Relation Aggregator

The Relation Aggregator operator takes as input the output of a relation, treats that output as if it were a database table, and performs a SQL SELECT on that table. The Relation Aggregator operator issues output in the form of a relation.

Use the Relation Aggregator operator when you want to perform SQL operations on a relation.

▼ To Create a Relation Aggregator Operator

1. **Drag a Relation Aggregator operator from the Palette to the Design view.**
2. **Double-click the Relation Aggregator operator.**

   The property editor opens with the default name of the Relation Aggregator operator and the output schema name populated. The Property Window displays the schema of the data that is input to the Relation Aggregator operator.
3. **In the Select field, specify attributes from which to select.**

   Any attribute that you specify must appear in the group-by clause. Attributes that you select as expression entries must be in the form of an attribute name, a literal, or an aggregate function supported by the database you use. Examples include COUNT, MAX, MIN, and AVG.
4. **In the Expression field, provide an SQL expression to further delimit your Select statement.**

   To save some typing, you can drag input attribute field names from the Inputs area into the Expression field.
5. **Optionally specify a Where statement in the Where field to provide a search condition, which cannot have a subquery.**
6. **In the group by field, specify a comma-separated list of attribute names, indicating how you want to group the attributes in the relation that is output from this operator.**
7. **Click OK.**

Time Based Aggregator

The Time Based Aggregator operator performs statistical analysis within a specified amount of time that you provide as a size, which is period of time over which you can perform a calculation, or the time slot. Increment specifies the frequency of the calculation; that is, how often you calculate the statistical analysis.

Assume that you want to calculate a stock price's 20–day moving average. You can supply a size of 20 in the property editor, and an increment that specifies how often you want to perform that calculation (for example, once a day).

Statistics that you can compute via SQL statements in the property editor of the Time Based Aggregator operator include:

- Sum
- Average
- Minimum
- Maximum
- COUNT

Use the Time Based Aggregator operator when you want to perform real-time statistical analysis. You can do simple or complex SQL manipulation within the time frame that you specify, by using the Select, From and Where clauses, as indicated in the property editor.

For example, given a stream of transactions of a stock, you can compute the new stream that holds the hourly minimum average and the maximum of the stock price.

▼ To Create a Time Based Aggregator Operator

1. **Drag a Time Based Aggregator operator from the Palette to the Design view.**
2. **Double-click the Time Based Aggregator operator.**

   The property editor opens.
3. **In the Start field, enter the time to start calculating the tasks that are to be performed by the Time Based Aggregator within the process.**
4. **In the Increment field, enter the time increment for the Time Based Aggregator to perform the analysis.**
5. **In the Size field, enter the time range for the Time Based Aggregator to perform the analysis.**
6. **In the Expression box, enter the expression for the SQL SELECT statement to specify the input attribute upon which the SELECT is performed. Add attribute names, data types. and sizes, as applicable.**

7 **In the From box, define the input on which to perform the selection.**

8 **In the Where clause box, provide additional filtering on records.**

For example:

```
WHERE price > 30.00 AND stockDate < '2006-01-01' ;
```

9 **In the Group by box, group a result into subsets that have matching values for one or more columns of the database by specifying a comma-separated list of qualified attribute names.**

10 **Click OK.**

Tuple Based Aggregator

The Tuple Based Aggregator operator performs statistical analysis for a specified number of records (also called tuples) that you provide as a size, and also for an increment that indicates how often you want the operation performed.

Statistics that you can compute via SQL statements in the property editor of the Tuple Based Aggregator operator include:

- Sum
- Average
- Minimum
- Maximum
- COUNT

Use the Tuple Based Aggregator operator when you want to perform statistical analysis on a specified number of tuples.

For example, provided a stream of stock transactions, the Tuple Based Aggregator operator computes a new stream that holds the minimum, average, and maximum of the stock price of every 10 transactions, in which the size is 10.

▼ To Create a Tuple Based Aggregator Operator

1 **Drag a Tuple Based Aggregator operator from the Palette to the Design view.**

2 **Double-click the Tuple Based Aggregator operator.**

The property editor opens.

3 **In the Start field, enter the record in which you want the operator calculations to begin.**

4 **In the Increment field, enter how often you want the analysis performed.**

5. **In the Expression box, enter the expression for the SQL SELECT statement that is used to specify the input attribute upon which the SELECT is performed. You can add, delete, or move attributes.**
6. **In the From box, define the input on which to perform the selection.**
7. **In the Where clause box, provide additional filtering on records.**

   For example:

   ```
   WHERE price > 30.00 AND stockDate < '2006-01-01' ;
   ```

8. **In the Group by box, group a result into subsets that have matching values for one or more columns of the database by specifying a comma-separated list of qualified attribute names.**
9. **Click OK.**

Correlation and Filter Operators

Correlation enables you to obtain data based on the relationship of two pieces of existing data. Filtering enables you to provide information to obtain a subset of data you want.

The following table lists the input and output for each operator.

Operator	Input	Output
Relation Map	Relation	Relation
Stream Projection and Filter	Stream	Stream
Tuple Serial Correlation	Stream	Stream

Relation Map

The Relation Map operator performs a select on two or more incoming relations, equivalent to a SQL join view of a minimum of one relation and additional tables and relations. The Relation Map operator can take multiple inputs.

You can use a Relation Map to join input from other operator sources, for example, from two or more tuple based windows, or from two or more partitioned windows.

For example, with the latest two-hour window of stock transactions and the latest two-hour window of trader information as input, you can compute the latest two-hour window of possible trades by joining the trader's name with the name provided in the trader information.

▼ To Create a Relation Map Operator

1. **Drag a Relation Map operator from the Palette to the Design view.**
2. **Connect at least two inputs to the Relation Map operator.**
3. **Double-click the Relation Map operator.**
   The property editor opens.
4. **In the Expression box, enter the expression for the SQL SELECT statement. Add attribute names, data types and sizes, as applicable. Add, delete, or move attributes.**
5. **In the From field, define the input on which to perform the selection.**
6. **In the Where clause field, provide additional search criteria.**
7. **Click OK.**

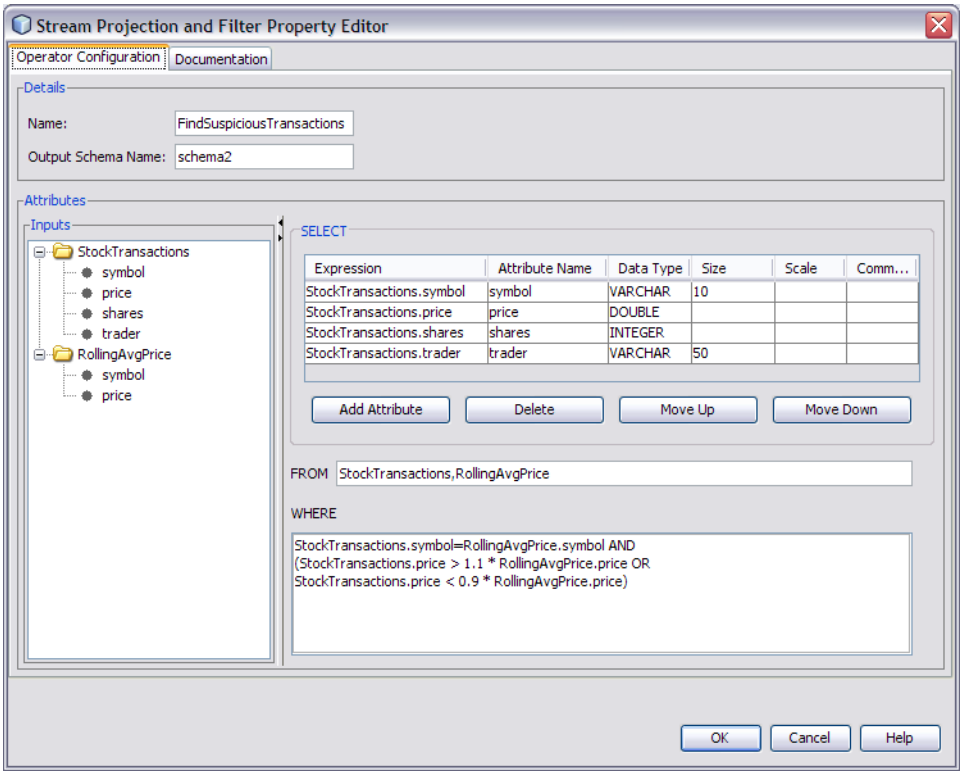
Stream Projection and Filter

The Stream Projection and Filter operator enables you to join a stream with multiple relations and tables, in order to create new events or to filter existing events based on specified conditions.

The input to the Stream Projection and Filter operator must include one (and only one) stream. The input can also include any number of relations and tables.

The output from the Stream Projection and Filter operator is a stream.

Configuring the Stream Projection and Filter operator resembles the process of writing a SQL SELECT statement. The property editor includes a SELECT area, a FROM area, and a WHERE area. The property editor also includes a tree structure that contains the input operators and their attributes.



In the SELECT area, you populate a table with one or more attributes. You can quickly add an existing attribute by selecting the attribute in the tree structure and dragging it to the Expression column. You can create a new attribute by using a mathematical computation in the Expression column. In the following example, two existing attributes are multiplied to create the new attribute.

```
total=Products.price*Products.tax
```

You can remove an attribute from the input stream by not including it in the SELECT area.

In the WHERE area, you can specify a search condition in the form of a boolean value expression. The boolean value expression can include the following types of predicates: comparison, between, in, like, null, quantified comparison, and exists.

One of the IEP tutorials uses the Stream Projection and Filter operator to check whether the price of each stock transaction is significantly different from the average price of all stock transactions during the last two minutes. The operator contains the following WHERE clause:

```
StockTransactions.symbol=RollingAvgPrice.symbol AND
(StockTransactions.price > 1.1 * RollingAvgPrice.price OR
```

```
StockTransactions.price < 0.9 * RollingAvgPrice.price)
```

▼ To Create a Stream Projection and Filter Operator

1. **Drag a Stream Projection and Filter operator from the Palette to the Design view.**
2. **Connect the output of an operator that has stream output to the input of the Stream Projection and Filter operator.**
3. **(Optional) Connect the output of one or more operators that have relation output to the input of the Stream Projection and Filter operator.**
4. **(Optional) Connect the output of one or more operators that have table output to the input of the Stream Projection and Filter operator.**
5. **Double-click the Stream Projection and Filter operator.**

   The property editor opens.
6. **If you want to change the default name, then change the value in the Name field.**
7. **In the SELECT area, populate the table with one or more attributes.**

   By default, the SELECT area contains only one attribute row. To create additional rows, click Add Attribute.

   You can drag and drop attributes from the tree structure into the Expression column. If you use this approach, then the operator name is automatically added to the FROM area.
8. **In the FROM area, specify the names of one or more input operators.**

   You can drag and drop operators from the tree structure into the FROM area.

   You can specify an alias for a table (for example, `TableInput0 t`) and then use the alias in the WHERE clause. You cannot specify an alias for a stream or a relation.

   If you specify more than one operator name, then separate the names with a comma (for example, `StockTransactions,RollingAvgPrice`).
9. **(Optional) In the WHERE area, specify a search condition in the form of a boolean value expression.**

   You can drag and drop operators and attributes from the tree structure into the WHERE area.
10. **Click OK.**

Tuple Serial Correlation

The Tuple Serial Correlation operator provides correlation of sequential events; that is, the operator uses a specification of a number of events from an existing stream to create a stream that contains larger events.

Assume that you have a stream of transactions of a stock, and you want to compute a new stream in which each event is composed of three consecutive events from the original stream. Using the latest two stock prices as output, you can build a trading model that predicts the next stock prices.

▼ To Create a Tuple Serial Correlation Operator

1. **Drag a Tuple Serial Correlation operator from the Palette to the Design view.**
2. **Double-click the Tuple Serial Correlation operator.**
   The property editor opens.
3. **In the Increment field, enter how often you want to obtain events via the Tuple Serial Correlation operator, based on the number of events.**
4. **In the Size field, specify the number of small consecutive events that you want to add to the larger cumulative event.**
5. **Click OK.**

Input Operators

Input operators enable you to receive data from external sources.

External Table Polling Stream

The External Table Polling Stream operator enables you to retrieve records from an external database table at a specified interval, and to output the records as a stream. The Properties Editor includes a wizard that you use to specify the various properties, including the table columns, the polling interval, and the number of records to retrieve.

After using the wizard, you can update the values for most of the properties from the property editor. However, if you want to add or remove table columns, then you must go through the wizard again.

In the event process, you can link the output of the External Table Polling Stream operator to more than one operator.

One of the properties is the JNDI name of the database resource. Before you deploy the event process, ensure that the JNDI name is configured in the application server to point to a valid database connection. For detailed instructions on configuring JNDI names, see the application server documentation.

▼ To Create an External Table Polling Stream Operator

1. **Drag an External Table Polling Stream operator from the Palette to the Design view.**
2. **Double-click the External Table Polling Stream operator.**
   The property editor opens.
3. **Click Select Table.**
   The Select External Table To Poll wizard appears.
4. **In step 1 of the wizard, do the following:**
   a. **Select the data source.**

      Note – Data sources are configured in the Databases node of the Services window.

   b. **Select the table from which you want to poll.**
   c. **Click Next.**
5. **In step 2 of the wizard, do the following:**
   a. **Select the column or columns that you want to retrieve from the table.**
   b. **(Optional) Add a condition to the Where Clause area (for example,** `amount > 100`**).**
   c. **Click Next.**
6. **In step 3 of the wizard, do the following:**
   a. **The wizard displays the column or columns that you specified in step 2. Select the one or more columns that uniquely identify each record.**

      Note – The operator uses this information to keep track of the last record that was retrieved. If you do not select any of the columns, then each fetch will start at the beginning of the table.

b. **In the Interval field and drop-down list, specify how often you want to retrieve records from the table.**

c. **In the Record Size field, specify the number of records to retrieve each time.**

d. **If you want to delete the records from the table after they have been fetched, then select the Delete Records check box.**

Note – If you do not select the one or more columns that uniquely identify each record, then selecting the Delete Records check box will ensure that you do not keep retrieving the same records.

---

e. **In the JNDI Name field, type the JNDI name of the database resource (for example,** `jdbc/iepseDerbyNonXA`**).**

f. **Click Finish. The property editor displays the values that you specified.**

7 **By default, the Preserve Last Fetched Record check box is selected. This setting indicates that if the event process is redeployed, then the operator starts retrieving records where it left off.**

If you leave the Preserve Last Fetched Record check box selected, then you must specify a table name in the Last Fetched Record Table field. This table will be created in the IEP database, not in the external database table.

If you want the operator to go back to the beginning of the table instead, then clear the Preserve Last Fetched Record check box.

8 **If you are retrieving records from an Oracle external database, then examine the Data Type column. If a data type is not one of the supported IEP data types, then change the data type to a supported IEP data type. For example, change the DECIMAL data type to the INTEGER data type.**

9 **(Optional) In the SELECT area, modify any of the default expressions by adding a SQL function.**

10 **(Optional) Click the Documentation tab and supply notes on this component.**

11 **Click OK.**

Replay Stream

The Save Stream and Replay Stream operators are intended to help you perform diagnostics. For example, you can use these operators when the output from an IEP Module project is not the expected output.

You first enable the Save Stream operator to begin saving an input stream to a database table. You then use the Replay Stream operator to replay the events that the Save Stream operator saved to the database table.

▼ To Create a Replay Stream Operator

1. **Drag a Replay Stream operator from the Palette to the Design view.**
2. **Connect the output of the Replay Stream operator to any operator that accepts stream input (for example, the Stream Output operator).**
3. **Double-click the Replay Stream operator.**
   The property editor opens.
4. **Click Select Table.**
   The Select Table which has Stream Events wizard appears.
5. **In step 1 of the wizard, do the following:**
   a. **Select the data source.**

   Note – Data sources are configured in the Databases node of the Services window.

   b. **Select the table where the events are being saved.**
   c. **Click Next.**
6. **In step 2 of the wizard, do the following:**
   a. **Select the column or columns that you want to replay.**
   b. **(Optional) Add a condition to the Where Clause area (for example,** `amount > 100`**).**
   c. **Click Next.**
7. **In step 3 of the wizard, do the following:**
   a. **Select the** `EMS_TIMESTAMP` **column.**
   b. **In the JNDI Name field, type the JNDI name of the database resource (for example,** `jdbc/iepseDerbyNonXA`**).**
   c. **Click Finish. The property editor displays the values that you specified.**

8 **By default, the Preserve Last Fetched Record check box is selected. This setting indicates that if the event process is redeployed, then the operator starts retrieving records where it left off.**

If you leave the Preserve Last Fetched Record check box selected, then you must specify a table name in the Last Fetched Record Table field. This table will be created in the IEP database, not in the external database table.

If you want the operator to go back to the beginning of the table instead, then clear the Preserve Last Fetched Record check box.

9 **(Optional) Click the Documentation tab and supply notes on this component.**

10 **Click OK.**

The Replay Stream operator reads events from the database table and sends the events to the next operator.

Stream Input

The Stream Input operator enables you to convert incoming messages to a format that can be used in the event process.

In the property editor, you define a schema that consists of one or more attributes. For example, the schema could consist of two attributes: stock symbol and stock price. Each attribute has a data type, such as INTEGER or VARCHAR.

At runtime, the event process reads the incoming messages from the Normalized Message Router and creates a stream based on the schema that you defined.

You can connect the Stream Input operator to any operator that accepts stream input (for example, the Time Based Window operator).

▼ To Create a Stream Input Operator

1 **Drag a Stream Input operator from the Palette to the Design view.**

2 **Double-click the Stream Input operator.**

The property editor opens.

3 **If you want to change the default name, then change the value in the Name field.**

4 **If you want to define the schema by adding attributes, then do the following:**

a. **For each attribute in the schema, click Add Attribute and specify the name, data type, size (if applicable), and scale (if applicable). You can optionally enter a comment.**

b. **To remove an attribute from the schema, select the attribute and click Delete.**

c. **To move an attribute up or down in the schema, select the attribute and click Move Up or Move Down.**

5 **If you want to define the schema by selecting an XML schema definition, then do the following:**

a. **Click Select Schema.**

b. **Select the element or type.**

c. **Click OK.**

6 **(Optional) Click the Documentation tab and supply notes on this component.**

7 **Click OK.**

Table Input

The Table Input operator enables you to use a relational database table as input to the event process.

You can connect the Table Input operator to any operator that accepts stream input (for example, the Time Based Window operator).

▼ To Create a Table Input Operator

1 **Drag a Table Input operator from the Palette to the Design view.**

2 **Double-click the Table Input operator.**

The property editor opens.

The Details section of the property editor displays the name of the operator and the output schema.

3 **In the Attributes section, you can add or delete attributes for this table input, or move the attributes up or down in order.**

4 **In the Data Type column in the Attributes section, select the appropriate data type from the list of data types.**

5 **In the Size column, you can specify the size of the data type if, for example, you specify the VARCHAR data type.**

6 **Click OK.**

Output Operators

Output operators enable you to send data from an event process to an external source.

Batched Stream Output

The Batched Stream Output operator enables you to output events in batches, rather than one event at a time.

Depending on the downstream JBI component, this approach can improve performance. For example, if you are using IEP with the File Binding Component, sending 10 events at a time might be faster than sending one event at a time.

▼ To Create a Batched Stream Output Operator

1 **Drag a Batched Stream Output operator from the Palette to the Design view.**

2 **Connect the input of the new Batched Stream Output component to an operator that has stream output.**

3 **Double-click the Batched Stream Output operator.**

The property editor opens. Notice that the component has inherited the schema of its input stream.

4 **If you want to include a timestamp on the output, then select the Include Timestamp check box.**

5 **In the Batch Size field, specify the number of events in a batch.**

6 **In the Maximum Delay field and drop-down list, specify the maximum amount of time that the operator will wait before sending a batch.**

For example, assume that the batch size is 10 and the maximum delay is 30 seconds. If 30 seconds have passed since the previous batch was sent, and only eight events have arrived, then the operator sends the eight events.

7 **(Optional) Click the Documentation tab and supply notes on this component.**

8 **Click OK.**

Invoke Stream

The Invoke Stream operator enables you to send a stream of data from one event process directly to another event process. The stream does not go through the Normalized Message Router.

For example, you could create two event processes:

- The first event process contains a Stream Input operator and a Stream Output operator.
- The second event process contains a Stream Input operator and an Invoke Stream operator.

You could then configure the Invoke Stream operator in the second event process to send data to the first event process.

▼ To Create an Invoke Stream Operator

1. **Create an event process that will receive a stream of data from another event process. This event process must contain a Stream Input operator.**
2. **Create an event process that will send a stream of data to the first event process. This event process must contain an Invoke Stream operator.**
3. **Double-click the Invoke Stream operator in the second event process.**

   The property editor opens.
4. **Click the ellipsis.**

   The Select Stream Input dialog box appears.
5. **Expand the node that represents the first event process and select the Stream Input operator.**
6. **Click OK to close the Select Stream Input dialog box.**
7. **The Inputs area contains the attributes of the second event process. Drag and drop attributes from the Inputs area to the Expression column in the SELECT area.**
8. **(Optional) Click the Documentation tab and supply notes on this component.**
9. **Click OK to close the property editor.**

   Now you can deploy both event processes. When an event is sent to the stream input of the second event process, the Invoke Stream operator sends the event to the stream input of the first event process.

Relation Output

The Relation Output operator sends out groups of tuples that reflect its input. When an event triggers a change in a relation result, the operator sends out changes rather than an entire result each time a triggering event is received.

The Relation Output operator sends out groups of tuples that reflect changes in the relation result. Whenever an event triggers a change in the relation result, the operator sends out records with a tag on the end. If the tag is a plus sign (+), then the record has been added. If the tag is a minus sign (-), then the record has been deleted.

If a change occurs as a result of the triggering event such that a record changes, IEP sends a delete record matching the changed record. IEP sends an add record when there is an updated or new version of that record. The operator only sends out change events rather than repeatedly sending the entire result.

If a triggering event enters into the event process, but the result of the final relation from that event does not change, then the relation output is not triggered and data is not sent.

Use the Relation Output operator to provide a summary of data that has been added or deleted.

▼ To Create a Relation Output Operator

1. **Drag a Relation Output operator from the Palette to the Design view.**
2. **Double-click the Relation Output operator.**
   The property editor opens.
3. **If you want to include a timestamp on the output, then select the Include Timestamp check box.**
4. **Click OK.**

Save Stream

The Save Stream and Replay Stream operators are intended to help you perform diagnostics. For example, you can use these operators when the output from an IEP Module project is not the expected output.

You first enable the Save Stream operator to begin saving an input stream to a database table. You then use the Replay Stream operator to replay the events that the Save Stream operator saved to the database table.

Before you begin, you must deploy an event process that contains a Stream Input operator.

Note – If you know up front that you want to save an input stream to a database table, then you can use the Save Stream operator at design time. The property editor enables you to configure the same properties described in the following procedure.

▼ To Enable the Save Stream Operator Dynamically at Runtime

1. **Start the IEP monitoring and debugging tool. For detailed information about this tool, see the Open ESB wiki.**

2. **Run the `listIEP` command. You must specify the IEP service unit name as a parameter. You can find the service unit name by going to the Services window of the IDE and expanding the Servers node, the GlassFish V2 node, the JBI node, the Service Assemblies node, and the individual service assembly node.**

   The command returns a list of plan names. For example:

   ```
   [IEP] listIEP CompositeApp1-IepModule1
   test_iep
   ```

3. **Run the `listOperators` command. You must specify the plan name as a parameter.**

   The command returns a list of the operators in the event process. For example:

   ```
   [IEP] listOperators test_iep
   StreamInput0
   StreamOutput0
   ```

4. **Run the `addSaveStream` command. You must specify the following parameters: the plan name, the name of the Stream Input operator, the JNDI name of the database resource, and the name of the database table where the input stream will be saved. You can optionally specify a boolean parameter called Is Global, which indicates whether the table and its data are kept between deployments. By default, the Is Global parameter is set to `true`.**

   The command returns the name of the Save Stream operator that was added. For example:

   ```
   [IEP] addSaveStream test_iep StreamInput0 jdbc/iepseDerbyNonXA STOCKTRANSACTIONS false
   The result is StreamInput0SaveStream0
   ```

5. **The Save Stream operator begins saving the input stream to the database table. If the table name that you specified does not exist, then the table is created.**

▼ To Disable the Save Stream Operator Dynamically at Runtime

1. **Start the IEP monitoring and debugging tool. For detailed information about this tool, see the Open ESB wiki.**

2. **Run the `removeSaveStream` command. You must specify the following parameters: the plan name and the name of the Save Stream operator. For example:**

   ```
   [IEP] removeSaveStream test_iep StreamInput0SaveStream0
   ```

3. **The Save Stream operator stops saving the input stream to the database table.**

   If the Is Global parameter of the `addSaveStream` command was set to `true`, then the table and its data are not deleted. However, if you subsequently modify the schema of the stream input (for example, by adding a column) and re-enable the Save Stream operator, then the table and its data are deleted at that time.

   If the Is Global parameter of the `addSaveStream` command was set to `false`, then the table and its data are deleted.

Stream Output

The Stream Output operator enables you to convert a stream of events into outgoing messages that can be sent to the Normalized Message Router and received by any JBI component.

When the WSDL document for the event processor is automatically generated, IEP uses the Stream Output operator to define various WSDL elements. For example, if the Stream Output operator is called `SuspiciousTransactions`, then the WSDL document contains an operation called `SuspiciousTransactions` and a message called `SuspiciousTransactions_Msg`.

▼ To Create a Stream Output Operator

1. **Drag a Stream Output operator from the Palette to the Design view.**
2. **Connect the input of the Stream Output operator to an operator that has stream output.**
3. **Double-click the Stream Output operator.**

   The property editor opens. Notice that the component has inherited the schema of its input stream.

4. **If you want to change the default name, then change the value in the Name field.**
5. **If you want to include a timestamp on the output, then select the Include Timestamp check box.**
6. **(Optional) Click the Documentation tab and supply notes on this component.**
7. **Click OK.**

Table Output

The Table Output operator provides static output in the form of relational tables.

Use the Table Output operator to provide relational table output from an event processor.

▼ To Create a Table Output Operator

1. **Drag a Table Output operator from the Palette to the Design view.**
2. **Double-click the Table Output operator.**
   The property editor opens.
3. **If you want to share the table related to the Table Output with other IEP processes or other applications, then select the `isGlobal` parameter.**
   The table related to the Table Output is created if it does not exist. The table remains after the undeployment or redeployment of the IEP process that contains it. IEP generates an additional column for every table that is output.

Relation Converter Operators

Relation Converter operators change the records in a relation or the format of a relation, or they can take a snapshot of the current relation.

The following table lists the input and output for each operator.

Operator	Input	Output
Delete Stream	Relation	Stream
Insert Stream	Relation	Stream
Notification Stream	Relation	Stream
Relation Stream	Relation	Stream

Delete Stream

The Delete Stream operator converts the deleted records of a changed relation into a stream. The operator takes all records that are in a previous table but not in the current table, and puts them into a stream with a timestamp.

Use the Delete Stream operator when you need to place deleted records from a relation into a stream.

▼ To Create a Delete Stream Operator

1. **Drag a Delete Stream operator from the Palette to the Design view.**
2. **Connect the input side of the Delete Stream operator to a relation output.**
3. **Double-click the Delete Stream operator.**
   The property editor opens.
4. **The Details section of the property editor displays the name of the operator and the output schema.**
5. **The Attributes section provides a picture of the current state of a stream.**
6. **Click OK.**

Insert Stream

The Insert Stream operator converts a relation into a stream.

When the Insert Stream operation is triggered, all changed or new records are issued as output.

The Insert Stream operator passes new records as output. In contrast, the Relation Output operator issues individual records tagged with either a plus, meaning a new record, or minus, meaning a record that was issued previously but is no longer part of the relation result.

Use the Insert Stream operator when you want to pass new or changed records into the output stream.

▼ To Create an Insert Stream Operator

1. **Drag an Insert Stream operator from the Palette to the Design view.**
2. **Connect it to an operator with a relation output.**
3. **Double-click the Insert Stream operator.**
   The property editor opens.
4. **The Details section of the property editor displays the name of the operator and the output schema.**
5. **Click OK.**

Notification Stream

The Notification Stream operator takes a relation as input, and outputs a stream that consists of events whose presence is determined by a specific time interval.

Assume that the following conditions are true:

- The time interval of the Notification Stream operator is set to 1 minute.
- An event arrives at the input relation at 2:00 in the afternoon.

If the event is still in the relation at 2:01 in the afternoon, then the event is included in the output stream for the first time.

If the event is still in the relation at 2:02 in the afternoon, then the event is included in the output stream for the second time.

If the event is still in the relation at 2:03 in the afternoon, then the event is included in the output stream for the third time.

If the event is no longer in the relation at 2:04 in the afternoon, then the event is no longer included in the output stream.

The scenario in "Gap Window" on page 48 involves creating a relation that indicates which message is missing at any point in time. You could use the Notification Stream operator to create an output stream of resend requests for the missing messages. Connect the Gap Window operator to a Notification Stream operator, configure the time interval, and then send the output to a Stream Output operator.

▼ To Create a Notification Stream Operator

1. **Drag a Notification Stream operator from the Palette to the Design view.**
2. **Connect the input of the new Notification Stream component to an operator that has relation output.**
3. **Double-click the Notification Stream operator.**

   The property editor opens. Notice that the component has inherited the schema of its input relation.
4. **Use the Notify Every field and drop-down list to specify the time interval.**
5. **(Optional) Click the Documentation tab and supply notes on this component.**
6. **Click OK.**

Relation Stream

The Relation Stream operator converts a relation result from an input operator into a stream. It provides a summary of the differences between the two consecutive tables, and places a timestamp on each event in the diff and places the result into the output stream.

Use the Relation Stream operator when you need relation result information in stream.

▼ To Create a Relation Stream Operator

1. **Drag a Relation Stream operator from the Palette to the Design view.**
2. **If needed, double-click the Relation Stream operator to examine the Details and Attribute sections.**

Relation Operators

Relation operators enable you to perform addition and combination, subtraction, intersection, or unique identification operations on relations.

The following table lists the input and output for each operator.

Operator	Input	Output
Distinct	Relation	Relation
Intersect	Relation	Relation
Minus	Relation	Relation
Union	Relation	Relation
Union All	Relation	Relation

Distinct

The Distinct operator retains records input from a relation that are unique. If duplicate records exist, they will not be sent out from the Distinct operator. All attributes of records must match to be filtered with the Distinct operators.

You typically use the Distinct operator in conjunction with a relation output from another operator. The Distinct operator does not forward duplicate records from input to it. Typical operators whose input the Distinct operator uses include:

- Attribute-Based Window operator

- Partitioned Window operator
- Time Based Window operator
- Tuple-Based Window operator
- Relation Map operator
- Relation Aggregator operator

When you use the Distinct operator, any records provided must be a complete duplicate, meaning that all attributes have to match for the Distinct operator to fill out all records.

The Distinct operator works with other operators. For example, you can use the Distinct operator with a Tuple-Based Window operator to limit the number of tuples that are affected by the operator. Or you can use the Distinct operator with the Time-Based Window operator to keep records for only 30 seconds, but to filter out records if there are no duplicates.

▼ To Create a Distinct Operator

1. **Drag a Distinct operator from the Palette to the Design view.**
2. **Connect an input from an operator that has a relation result.**
3. **Double-click the Distinct operator.**

   The property editor opens with the name of the Distinct operator and the output schema name populated. The property editor displays the schema of the data that is input to the Distinct operator.
4. **(Optional) Click the Documentation tab and supply notes on this component.**
5. **Click OK.**

Intersect

The Intersect operator enables definition of relation intersection. In effect, the operator behaves like the SQL INTERSECT command and acts as an `AND` operator (that is, values are selected only if they appear in all inputs provided to the operator).

All input schemas to the Intersect operator must be identical in format: column name and type must match for all attributes. You cannot select a subset of columns (attributes) from the input schemas, as you can with the SQL INTERSECT command.

▼ To Create an Intersect Operator

1. **Drag an Intersect operator from the Palette to the Design view.**
2. **Connect two input relations to the Intersect operator.**
3. **Double-click the Intersect operator.**
   The property editor opens.
4. **Examine the Details and Attributes sections.**
5. **(Optional) Click the Documentation tab and supply notes on this component.**
6. **Click OK.**

Minus

The Minus operator subtracts one stream from another stream. The order of subtraction is determined by the order in which you connect the input relations. You cannot subtract unlike operator attributes. The input schemas must be identical, with identical names and types for all attributes.

Specify the operator to subtract from, then choose the expression in the Expression drop-down list, where you can change the order for the subtraction operation. As with other operators, you can specify multiple input streams to a minus operation, and provide filters before the Minus operator.

▼ To Create a Minus Operator

1. **Drag a Minus operator from the Palette to the Design view.**
2. **Connect two or more input relations to the Minus operator.**
3. **Double-click the Minus operator.**
   The Expression field provides the order of the minus expression. This is initially determined by the order in which the inputs were connected. You can determine this by opening the properties pane for the Minus operator. The input ID list provides the order of the operators that the Minus operator subtracts. Next, open the properties pane for the relations feeding into the Minus operator and examine their ID.
4. **To change the order in the Expression dialog box, choose the Order by drop-down list and a new primary operator. The latter becomes the first operator on the Expression field. You can connect**

additional inputs to the Minus operator at a later time. You can also remove inputs. When you remove inputs, the Expression is automatically updated.

Union

The Union operator combines elements from input relations. The Union operator captures one of every record in every relation and provides one of every record as output, with duplicates removed.

All input relations must have the same schema. Relations that are output from the Union operator will have the same schema.

Because the Union operator does not pass through duplicates, if two different files input to it indicate size of 1, the behavior is similar to having two single streams input to a merged file.

That is, for each event a new row is added, and one subtracted. Note that if the input files are identical (the rows are exactly the same), and the Tuple Size is also 1, then the previous output is deleted, but nothing is added.

▼ To Create a Union Operator

1. **Drag a Union operator from the Palette to the Design view.**
2. **Connect the input of the new Union component to two or more operators that have relation output.**
3. **Double-click the Union operator.**
   The property editor opens. Notice that the component has inherited the schema of its input relations.
4. **As needed, configure the following property: Name:**
5. **(Optional) Click the Documentation tab and supply notes on this component.**
6. **Click OK.**

Union All

The Union All operator combines elements from input relations, and captures one of every record in every relation and provides one of every record as output, including duplicates.

All input relations must have the same schema. Relations that are output from the Union All operator have the same schema.

You typically use the Union All operator to join data from two separate streams, such as when your application requires that you match data from one stream against that of another, without excluding duplicate records from the output.

▼ To Create a Union All Operator

1. **Drag a Union All operator from the Palette to the Design view.**
2. **Connect the input of the new Union All component to two or more operators that have relation output.**
3. **Double-click the Union All operator.**
   The property editor opens. Notice that the component has inherited the schema of its input relations.
4. **As needed, configure the following property: Name:**
5. **(Optional) Click the Documentation tab and supply notes on this component.**
6. **Click OK.**

Sequence Operators

Sequence operators enable you to order events based on attributes that have sequential order.

The following table lists the input and output for each operator.

Operator	Input	Output
Contiguous Order	Stream	Stream
Gap Window	Stream	Relation

Contiguous Order

The Contiguous Order operator defines a stream that consists of events selected by one or more attributes and then placed in sequence sorted by another attribute.

Assume that you have the following scenario:

- A server receives a stream of messages from various hardware devices.
- For each hardware device, the messages have unique IDs (1, 2, 3, 4, and so on).

- Because of an unreliable network, messages may be lost or received in a different order.
- For each hardware device, you want to sort the messages by their unique ID.

You could add a Stream Input operator with the following attributes:

- `deviceID` (VARCHAR)
- `msgID` (BIGINT)
- `msg` (VARCHAR)

You could then connect the Stream Input operator to a Contiguous Order operator.

In the property editor of the Contiguous Order operator, set the partition key to the `deviceID` attribute. This setting divides the input stream into multiple substreams, one per device.

In the property editor of the Contiguous Order operator, set the sort value to the `msgID` attribute and the start value to 1. For each substream, the Contiguous Order operator sorts the messages by their unique ID.

For example, let's say that the messages from one device arrive in the following order: 3, 2, 5, 1, 7, 4, 4, 6.

When message 1 is received, the Contiguous Order operator will output messages 1, 2, and 3. When message 4 is received, the Contiguous Order operator will output messages 4 and 5. When message 6 is received, the Contiguous Order operator will output messages 6 and 7.

The Contiguous Order operator ignores duplicate events. Therefore, the output stream in the hardware device scenario contains only one instance of message 4.

Note that in the output stream, the output from the multiple substreams is interspersed.

▼ To Create a Contiguous Order Operator:

1. **Drag a Contiguous Order operator from the Palette to the Design view.**
2. **Connect the input of the new Contiguous Order component to an operator that has stream output.**
3. **Double-click the Contiguous Order operator.**

   The property editor opens. Notice that the component has inherited the schema of its input stream.
4. **In the Attributes area, set the Partition Key column to the attribute or attributes that you want to use to divide the input stream into substreams.**
5. **In the Sort By drop-down list, select the attribute that you want to sort by.**

   The attribute must have values that can be sorted sequentially.

6. **In the Start field, type the attribute value that you want to start with.**
7. **(Optional) Click the Documentation tab and supply notes on this component.**
8. **Click OK.**

Gap Window

The Gap Window operator defines a relation that consists of events that are missing from the input stream, based on an attribute that has a sequential order.

Assume that you have the following scenario:

- A server receives a stream of messages from various hardware devices.
- For each hardware device, the messages have unique IDs (1, 2, 3, 4, and so on).
- Because of an unreliable network, messages may be lost or received in a different order.
- For each hardware device, you want to sort the messages by their unique ID.

You could add a Stream Input operator with the following attributes:

- `deviceID` (VARCHAR)
- `msgID` (BIGINT)
- `msg` (VARCHAR)

You could then connect the Stream Input operator to a Gap Window operator.

In the property editor of the Gap Window operator, set the partition key to the `deviceID` attribute. This setting divides the input stream into multiple substreams, one per device.

In the property editor of the Gap Window operator, set the sort value to the `msgID` attribute and the start value to 1. For each substream, the Gap Window operator creates an output relation that indicates which message is missing at any point in time.

For example, let's say that the messages from one device arrive in the following order: 3, 2, 5, 1, 7, 4, 4, 6.

When message 3 is received, the Gap Window operator will output message 1. When message 1 is received, the Gap Window operator will output message 4. When message 4 is received, the Gap Window operator will output message 6. When message 6 is received, the Gap Window operator will output message 8.

Note that in the output relation, the output from the multiple relations is interspersed.

▼ To Create a Gap Window Operator:

1. **Drag a Gap Window operator from the Palette to the Design view.**
2. **Connect the input of the new Gap Window component to an operator that has stream output.**
3. **Double-click the Gap Window operator.**
   The property editor opens. Notice that the component has inherited the schema of its input stream.
4. **In the Attributes area, set the partition key to the attribute or attributes that you want to use to divide the input stream into substreams.**
5. **In the Sort By drop-down list, select the attribute that you want to sort by.**
   The attribute must have values that can be sorted sequentially.
6. **In the Start field, type the attribute value that you want to start with.**
7. **(Optional) Click the Documentation tab and supply notes on this component.**
8. **Click OK.**

Stream Converter Operators

Stream Converter operators convert stream data formats into other formats, and can perform additional processing.

The following table lists the input and output for each operator.

Operator	Input	Output
Attribute Based Window	Stream	Relation
Partitioned Window	Stream	Relation
Time Based Window	Stream	Relation
Tuple Based Window	Stream	Relation

Attribute Based Window

The Attribute Based Window operator converts an input stream to a relation based on a specified attribute and a size that defines the range of values for the attribute.

Assume that you select an attribute that has the INTEGER data type and that you set the size to 10. If the most recently received event has a value of 50 for the attribute, then the relation consists of all events that have a value of between 40 and 50 for the attribute.

The following data types are supported: INTEGER, BIGINT, DOUBLE, DATE, and TIMESTAMP. You cannot select an attribute that has the VARCHAR data type.

The input to the Attribute Based Window operator is a stream.

The output from the Attribute Based Window operator is a relation.

▼ To Create an Attribute Based Window Operator

1. **Drag an Attribute Based Window operator from the Palette to the Design view.**
2. **Connect the output of an operator that has stream output to the input of the Attribute Based Window operator.**
3. **Double-click the Attribute Based Window operator.**
   The property editor opens. The attributes of the input schema are displayed in read-only mode.
4. **In the Attribute drop-down menu, select the attribute that you want to track.**
5. **In the Size field, enter a numerical value that the operator will use to define the range of values.**
   The operator determines the lowest value of the range by subtracting the size from the attribute value of the most recently received event.

   For attributes with the DATE data type, the size represents the number of days.

   For attributes with the TIMESTAMP data type, the size represents the number of seconds.
6. **(Optional) Click the Documentation tab and supply notes on this component.**
7. **Click OK.**

Partitioned Window

The Partitioned Window operator converts an input stream to a relation based on one or more attributes that serve as the partition key.

Assume that you have the following scenario:

- The input stream contains census data.
- You select the `LastName` attribute.
- You set the number of events to 2.

Each relation will consist of the last two events that contain the `LastName` attribute.

The input to the Partitioned Window operator is a stream.

The output from the Partitioned Window operator is a relation.

▼ To Create a Partitioned Window Operator

1. **Drag a Partitioned Window operator from the Palette to the Design View.**
2. **Connect the output of an operator that has stream output to the input of the Partitioned Window operator.**
3. **Double-click the Partitioned Window operator.**
   The property editor opens. The attributes of the input schema are displayed in read-only mode.
4. **In the Size field, specify the number of events.**
5. **In the Partition Key column, select one or more attributes.**
6. **(Optional) Click the Documentation tab and supply notes on this component.**
7. **Click OK.**

Time Based Window

The Time Based Window operator converts an input stream to a relation based on a specified period of time.

Assume that you set the period of time to five seconds. Each relation will consist of all events that arrived in the last five seconds.

The input to the Time Based Window operator is a stream.

The output from the Time Based Window operator is a relation.

You can combine the Time Based Window operator with a subsequent Relation Output operator to provide a timestamp record for the records retained.

▼ To Create a Time Based Window Operator

1. **Drag a Time Based Window operator from the Palette to the Design view.**
2. **Connect the output of an operator that has stream output to the input of the Time Based Window operator.**

3. **Double-click the Time Based Window operator.**
   The property editor opens. The attributes of the input schema are displayed in read-only mode.
4. **In the Size field and drop-down menu, specify the period of time.**
5. **(Optional) Click the Documentation tab and supply notes on this component.**
6. **Click OK.**

Tuple Based Window

The Tuple Based Window operator converts an input stream to a relation based on a specified number of events.

Assume that you set the number of events to two. Each relation will consist of the most recent two events.

The input to the Tuple Based Window operator is a stream.

The output from the Tuple Based Window operator is a relation.

▼ To Create a Tuple Based Window Operator

1. **Drag a Tuple Based Window operator from the Palette to the Design view.**
2. **Connect the output of an operator that has stream output to the input of the Tuple Based Window operator.**
3. **Double-click the Tuple Based Window operator.**
   The property editor opens. The attributes of the input schema are displayed in read-only mode.
4. **In the Size field, specify the number of events.**
5. **(Optional) Click the Documentation tab and supply notes on this component.**
6. **Click OK.**

WSDL Documents in IEP Module Projects

When you save an IEP Module project, IEP generates a Web Services Description Language (WSDL) document for each event processor.

The WSDL documents contain the endpoints for the event processors.

Caution – Do not edit any section of the generated WSDL other than the following sections:

- HTTP SOAP binding and service generation for input
- File binding and service generation for output

IEP generates a WSDL document according to specific rules based on the XML Schema (the XSD) that are important to understand. These rules are provided as reference and to enable you to change specific binding components and service engines that can be changed.

The name of the WSDL document is derived from the name of the event processor. For example, if the event processor is called `insiderTrade.iep`, then the WSDL document is called `insiderTrade.wsdl`.

Note – If you subsequently rename the event processor, IEP does not refactor the name of the WSDL document.

The target namespace is a convention of an XML Schema that enables the WSDL document to refer to itself.

In the process of representing the event process in the WSDL document, the name of your event process is concatenated with the string `_iep` to form the target namespace. For example, if the name of the event process is `MyEventProcessor`, then the target namespace of the WSDL is `MyEventProcessor_iep`.

```
<definitions targetNamespace="MyEventProcessor_iep"
             xmlns:tns="MyEventProcessor_iep"
             xmlns:typens="MyEventProcessor_iep"
             xmlns:defns="MyEventProcessor_iep"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:file="http://schemas.sun.com/jbi/wsdl-extensions/file/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Data Types in the WSDL Document

The WSDL document defines XSD types (data types). IEP data types, as observable and editable from the property editor for a specific operator, have their counterparts in XSD data types.

For each operator, IEP generates two XSD elements: a regular message object and a batch message object.

Each XSD element contains the following:

- Operator name
- Attribute name for each attribute in the attribute list
- XSD data type

Attributes have specific data types and have their equivalent representation in the XSD.

IEP Data Type	XSD Data Type
INTEGER	int
BIGINT	long
DOUBLE	double
VARCHAR	string
DATE	date
TIMESTAMP	dateTime

The following types are based on a simple event process that contains a Stream Input operator and a Stream Output operator.

```
<types>
   <xsd:schema targetNamespace="MyEventProcessor_iep"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="MyStreamInput_MsgObj">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="AttrInteger" type="xsd:int"/>
               <xsd:element name="AttrBigInt" type="xsd:long"/>
               <xsd:element name="AttrDouble" type="xsd:double"/>
               <xsd:element name="AttrVarchar" type="xsd:string"/>
               <xsd:element name="AttrDate" type="xsd:date"/>
               <xsd:element name="AttrTimestamp" type="xsd:dateTime"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="MyStreamInputBatch_MsgObj">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="MyStreamInput_MsgObj" minOccurs="0"
maxOccurs="unbounded">
                  <xsd:complexType>
```

```
                  <xsd:sequence>
               <xsd:element name="AttrInteger" type="xsd:int"/>
               <xsd:element name="AttrBigInt" type="xsd:long"/>
               <xsd:element name="AttrDouble" type="xsd:double"/>
               <xsd:element name="AttrVarchar" type="xsd:string"/>
               <xsd:element name="AttrDate" type="xsd:date"/>
               <xsd:element name="AttrTimestamp" type="xsd:dateTime"/>
                  </xsd:sequence>
               </xsd:complexType>
               </xsd:element>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="StreamOutput0_MsgObj">
         <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="AttrInteger" type="xsd:int"/>
              <xsd:element name="AttrBigInt" type="xsd:long"/>
              <xsd:element name="AttrDouble" type="xsd:double"/>
              <xsd:element name="AttrVarchar" type="xsd:string"/>
              <xsd:element name="AttrDate" type="xsd:date"/>
              <xsd:element name="AttrTimestamp" type="xsd:dateTime"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
   </xsd:schema>
</types>
```

Message Objects in the WSDL Document

The WSDL document contains two types of message objects:

- Regular message object
- Batch message object

IEP uses the regular message object when you are sending in one message at a time. A batch message object is useful when you want to send in multiple messages.

```
<message name="MyStreamInput_Msg">
   <part name="input" element="typens:MyStreamInput_MsgObj"/>
</message>
<message name="MyStreamInputBatch_Msg">
   <part name="input" element="typens:MyStreamInputBatch_MsgObj"/>
</message>
<message name="StreamOutput0_Msg">
   <part name="output" element="typens:StreamOutput0_MsgObj"/>
</message>
```

If one of the messages in the batch message group fails to be processed appropriately in the IEP, others will fail as well.

With a single message object, message failure is on a one-by-one basis.

Bindings and Services in the WSDL Document

By default, the WSDL documents that are automatically generated in an IEP Module project include concrete information (bindings and services).

You can configure IEP to generate abstract WSDL documents instead.

Generating Concrete WSDL Documents

By default, the WSDL documents that are automatically generated in an IEP Module project include concrete information (bindings and services).

If the event processor has a Stream Input operator, then the WSDL document contains a SOAP-based binding and service.

If the event processor has a Stream Output operator, a Batched Stream Output operator, or a Relation Output operator, then the WSDL document contains a file-based binding and service.

The following example shows a SOAP-based binding and service, followed by a file-based binding and service.

```
<binding name="InputBinding" type="defns:InputPt">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="MyStreamInput">
      <soap:operation soapAction="MyStreamInput"/>
      <input>
         <soap:body use="literal"/>
      </input>
   </operation>
   <operation name="MyStreamInputBatch">
      <soap:operation soapAction="MyStreamInputBatch"/>
      <input>
         <soap:body use="literal"/>
      </input>
   </operation>
</binding>

<service name="InputService">
   <port name="InputPort" binding="tns:InputBinding">
      <soap:address location="http://localhost:12100/service/newEventProcessor_iep"/>
   </port>
```

```
</service>

<binding name="OutputBinding_StreamOutput0" type="defns:OutputPt_StreamOutput0">
   <file:binding/>
   <operation name="StreamOutput0">
      <file:operation/>
      <input>
         <file:message fileName="StreamOutput0.txt"
                 fileNameIsPattern="false"
                 addEOL="false"
                 multipleRecordsPerFile="true"
                 use="literal"/>
      </input>
   </operation>
</binding>

<service name="OutputService_StreamOutput0">
   <port name="OutputPort_StreamOutput0" binding="tns:OutputBinding_StreamOutput0">
      <file:address fileDirectory="C:/temp/newEventProcessor_iep"/>
   </port>
</service>
```

You might need to manually update the default settings for any binding and service. For example, the default file directory is a Windows path, which would not work correctly on a UNIX system.

For detailed information about the binding and service configuration, see the documentation for the HTTP Binding Component and the File Binding Component.

Note – If you manually update the WSDL document and then save the IEP Module project again, the changes that you made to the WSDL document are overwritten.

Generating Abstract WSDL Documents

You can configure IEP to generate abstract WSDL documents by editing a flag in the IEP project properties file. The bindings and services are not included.

With this approach, you can define the bindings and services by using the Composite Application Service Assembly (CASA) Editor.

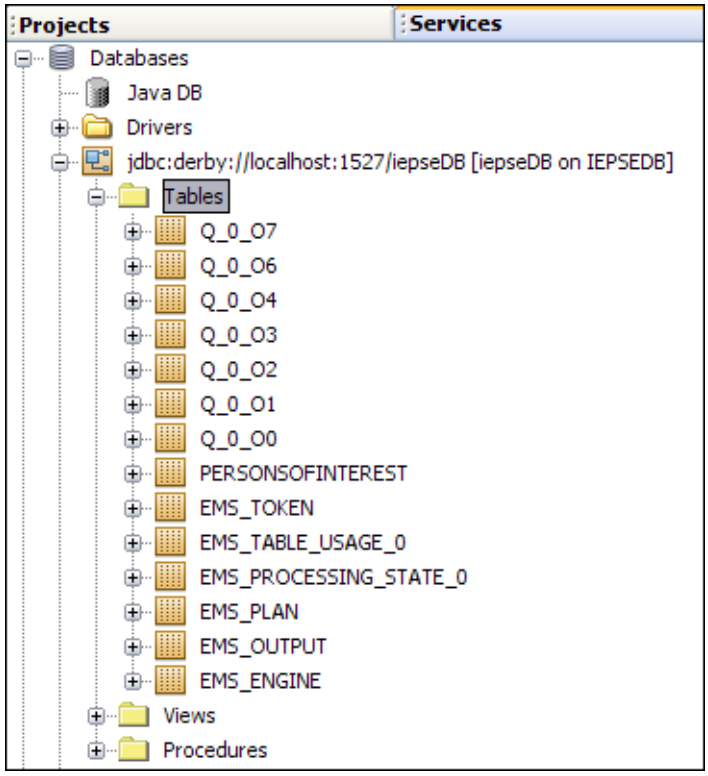
Caution – If you use the CASA Editor, do not clone the IEP WSDL document to customize its generated bindings and services. Updates to the generated WSDL documents will not be updated after the cloning. Over time, the cloned and edited WSDL document deployed for the bindings will become inconsistent with the WSDL document deployed for the IEP Service Engine.

▼ To Generate Abstract WSDL Documents

1. **Go to the location of the IEP Module project and open the `project.properties` file.**
2. **Change the value of the `always.generate.abstract.wsdl` flag to `true`.**
3. **Save the `project.properties` file.**

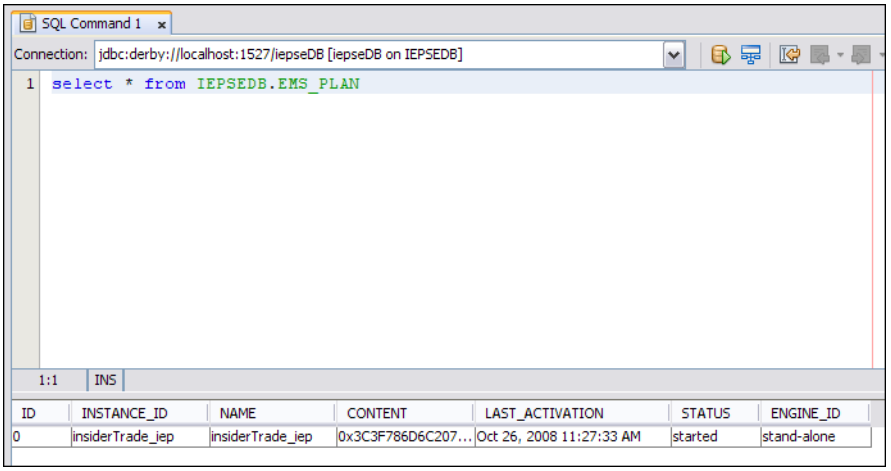
Understanding the IEP Database

IEP uses a set of database tables to maintain information about the IEP Service Engine and deployed event processes. You can connect to the database through NetBeans, and then view the tables and their content through NetBeans windows. To view the IEP database tables, click the Services window of the NetBeans IDE. Expand the Databases node, the connection node, and the Tables node.



To view the contents of an IEP database table, right-click the table node and choose View Data. The SQL Editor appears and displays the appropriate SQL SELECT statement, which is performed on the table. The results of the query appear near the bottom of the SQL Editor.

The following image shows the contents of the EMS_PLAN table.



You can use any of the following database servers for the IEP database:

- Java DB (bundled with GlassFish)
- Oracle Database 11*g*
- Oracle Database 10*g*
- Oracle 9*i* Database

Configuring the IEP Database to Use Oracle

By default, the IEP database is configured to use Java DB. Java DB is Sun's supported distribution of the open source Apache Derby database. After installation, you can configure the IEP database to use Oracle instead.

The first task is to create an IEP user in the Oracle database. You can run a provided SQL script that creates the IEP user and grants the appropriate privileges. The default version of the script contains the following SQL statements:

```
CREATE TABLESPACE "IEPSEDB_DB"
 DATAFILE
  'IEPSEDB_DB1.dat' SIZE 2000M,
  'IEPSEDB_DB2.dat' SIZE 2000M;

CREATE USER IEPSEDB IDENTIFIED BY IEPSEDB
DEFAULT TABLESPACE IEPSEDB_DB
QUOTA UNLIMITED ON IEPSEDB_DB
TEMPORARY TABLESPACE temp
QUOTA 0M ON system;

GRANT CONNECT TO IEPSEDB;
GRANT RESOURCE TO IEPSEDB;
```

```
GRANT CREATE VIEW TO IEPSEDB;

grant select on sys.dba_pending_transactions to IEPSEDB;
grant select on sys.pending_trans$ to IEPSEDB;
grant select on sys.dba_2pc_pending to IEPSEDB;
grant execute on sys.dbms_system to IEPSEDB;
grant select on SYS.dba_2pc_neighbors to IEPSEDB;
grant force any transaction to IEPSEDB;
```

You need to create two sets of connection pools and JDBC resources. A *connection pool* is a group of reusable connections for a particular database. The server maintains a pool of available connections to increase performance. A *JDBC resource* provides applications with a means of connecting to a database. Additional configuration of the service engine is also required.

To configure the IEP Service Engine to use Oracle, perform the following steps:

▼ To Create the IEP User in the Oracle Database

1. **In a web browser, enter the following URL:** `http://wiki.open-esb.java.net/Wiki.jsp?page=HowToRunIEPOnOracle`**.**
2. **Download the** `create_iepse_user.sql` **script.**
3. **Open the** `create_iepse_user.sql` **script and review the instructions.**
4. **The names of the tablespace, data files, user, and password include the string** `IEPSEDB`**. If you want to change this string to a different string, then replace all occurrences of** `IEPSEDB` **with the new string.**
5. **Connect to the Oracle database as a user with the** `SYSDBA` **privilege.**
6. **Run the** `create_iepse_user.sql` **script against the database.**

▼ To Install the Oracle Database Driver in the Application Server

1. **Go to the computer where the application server is installed.**
2. **Copy and paste the Oracle database driver (for example,** `ojdbc14.jar`**) to the** *glassfish-home*`/lib` **directory.**
3. **Restart the application server.**

▼ To Create the Non-XA Connection Pool

1. **Log in to the GlassFish Admin Console.**
2. **In the left navigation panel, expand Resources and JDBC, and then select Connection Pools.**
3. **Click New.**
   The New JDBC Connection Pool page appears.
4. **Do the following:**
   a. **In the Name field, specify a name for the non-XA connection pool (for example,** `iepseOraclePoolNonXA`**).**
   b. **In the Resource Type field, select** `javax.sql.DataSource`**.**
   c. **In the Database Vendor field, select** `Oracle`**.**
   d. **Click Next.**
5. **Scroll down to the Additional Properties table, and then do the following:**
   a. **In the User row, enter the user name of the IEP user (for example,** `IEPSEDB`**).**
   b. **In the Password row, enter the password of the IEP user (for example,** `IEPSEDB`**).**
   c. **In the URL row, enter the string for connecting to the database (for example,** `jdbc:oracle:thin:@myserver:1521:orcl`**).**
6. **Click Finish.**
   The connection pool is created.
7. **Click the connection pool that you just created.**
   The Edit Connection Pool page appears.

8 **Click Ping.**

The Admin Console attempts to connect to the database. If the connection does not succeed, check to see whether the database is running and verify the database connectivity properties, such as the URL string.

▼ To Create the Non-XA JDBC Resource

1 **In the left navigation panel of the Admin Console, expand Resources and JDBC, and then select JDBC Resources.**

2 **Click New.**

The New JDBC Resource page appears.

3 **In the JNDI Name field, specify a unique name for the non-XA JDBC resource. By convention, the name begins with the `jdbc/` string. For example:**

```
jdbc/iepseOracleNonXA
```

You will use the JNDI name in a later procedure.

4 **In the Pool Name field, select the non-XA connection pool that you created in the previous procedure.**

5 **Click OK.**

The JDBC resource is created.

▼ To Create the XA Connection Pool

1 **In the left navigation panel of the Admin Console, expand Resources and JDBC, and then select Connection Pools.**

2 **Click New.**

The New JDBC Connection Pool page appears.

3 **For step 1 of the connection pool, do the following:**

a. **In the Name field, specify a name for the XA connection pool (for example, `iepseOraclePoolXA`).**

b. **In the Resource Type field, select `javax.sql.XADataSource`.**

c. **In the Database Vendor field, select `Oracle`.**

d. **Click Next.**

4. **For step 2 of the connection pool, do the following:**

   a. **Scroll down to the Connection Validation section.**

   b. **Select the Enabled check box that appears to the right of the Allow Non Component Callers label.**

   c. **Scroll down to the Additional Properties table.**

   d. **In the User row, enter the user name of the IEP user (for example,** `IEPSEDB`**).**

   e. **In the Password row, enter the password of the IEP user (for example,** `IEPSEDB`**).**

   f. **In the URL row, enter the string for connecting to the database (for example,** `jdbc:oracle:thin:@myserver:1521:orcl`**).**

5. **Click Finish.**

   The connection pool is created.

6. **Click the connection pool that you just created.**

   The Edit Connection Pool page appears.

7. **Click Ping.**

   The Admin Console attempts to connect to the database. If the connection does not succeed, check to see whether the database is running, and verify the connectivity parameters, such as the URL string.

▼ To Create the XA JDBC Resource

1. **In the left navigation panel of the Admin Console, expand Resources and JDBC, and then select JDBC Resources.**

2. **Click New.**

   The New JDBC Resource page appears.

3. **In the JNDI Name field, specify a unique name for the XA JDBC resource. By convention, the name begins with the** `jdbc/` **string. For example:**

   ```
   jdbc/iepseOracleXA
   ```

   You use the JNDI name in a later procedure.

4. **In the Pool Name drop-down menu, select the XA connection pool that you created.**

5 **Click OK.**

The JDBC resource is created.

▼ To Enable Automatic Recovery of XA Transactions

1 **In the left navigation panel of the Admin Console, expand Configuration and then select Transaction Service.**

2 **Select the Enabled check box that appears to the right of the On Restart label.**

3 **Click Save.**

▼ To Configure the IEP Service Engine to Use the JDBC Resources

1 **Log in to the NetBeans IDE.**

2 **In the Services window, expand Servers > GlassFish V2 > JBI > Service Engines.**

3 **If the IEP Service Engine is not started, right-click** `sun-iep-engine` **and select Start.**

4 **Right-click** `sun-iep-engine` **and select Properties.**

The Properties dialog box appears.

5 **In the Non XA Data Source Name property, enter the non-XA JDBC resource that you created (for example,** `jdbc/iepseOracleNonXA`**).**

6 **In the XA Data Source Name property, enter the XA JDBC resource that you created (for example,** `jdbc/iepseOracleXA`**).**

7 **In the Database Schema Name property, enter the user name of the IEP user (for example,** `IEPSEDB`**).**

In an Oracle database, the database schema name is identical to the user name.

8 **Click OK.**

▼ To Restart the IEP Service Engine and Create the Database Tables

When you restart the IEP Service Engine, the database tables are automatically created in the database you specified through the connection pools and runtime properties.

1 **If any Composite Application projects that contain event processes are currently deployed, undeploy the projects.**

2. **In the Services window, expand Servers > GlassFish V2 > JBI > Service Engines.**
3. **Right-click** `sun-iep-engine` **and select Stop.**
4. **Right-click** `sun-iep-engine` **and select Start.**
5. **If you undeployed any Composite Application projects, you can now redeploy the projects.**

IEP Service Engine-Specific Database Tables

When you start the IEP Service Engine for the first time, the following tables are created in the IEP database. These tables apply to the IEP Service Engine as a whole.

Caution – Do not delete or alter these tables.

EMS_PLAN Table

The EMS_PLAN table maintains information about the event processes that are deployed to the IEP Service Engine. The EMS_PLAN table contains the following columns:

- ID. An integer that uniquely identifies the event process.
- INSTANCE_ID. A value that represents the event process.
- NAME. The name of the event process.
- CONTENT. The XML content of the event process. The value is displayed in binary format.
- LAST_ACTIVATION. The last time the event process was started.
- STATUS. The status of the event process definition. The valid values are `started`, `stopped`, `deployed`, and `undeployed`.
- ENGINE_ID. The ID of the cluster node that is running the event process. If you are not running IEP in cluster mode, then the value is `stand-alone`.

EMS_OUTPUT Table

The EMS_OUTPUT table maintains information for event processes that have a Table Output operator. The EMS_OUTPUT table contains the following columns:

- PLAN_INSTANCE_ID. The instance ID of the event process that has the Table Output operator.
- OUTPUT_NAME. The name of the table.
- OUTPUT_DESCRIPTION. The description of the table.
- LAST_UPDATED. The last time the output table was updated.

EMS_ENGINE Table

The EMS_ENGINE table is used when you are running IEP in cluster mode. The EMS_ENGINE table contains the following columns:

- ID. A value that uniquely identifies the IEP instance (`instance1`, `instance2`, and so on). If you are not running IEP in cluster mode, then the value is `stand-alone`.
- INSTANCE_LOCATION. The IP address of the computer where the IEP instance is running.
- LEASE_EXPIRATION. A timestamp that the cluster nodes use to determine which IEP instance is active.

EMS_TOKEN Table

The EMS_TOKEN table is used when you are running IEP in cluster mode. The EMS_TOKEN table contains the following columns:

- ID. The unique identifier of a token that can be acquired and released by cluster nodes.
- NAME. The name of a token that can be acquired and released by cluster nodes.

Event Process-Specific Database Tables

When you deploy an event process, the following tables are created in the IEP database. If you undeploy the event process, then the tables are deleted.

Caution – Do not delete or alter these tables.

EMS_PROCESSING_STATE_*N* Tables

The EMS_PROCESSING_STATE_*N* tables are used by the IEP Service Engine to maintain execution state for an event process. The EMS_PROCESSING_STATE_*N* tables contain the following columns:

- PROCESSING_STATE. The execution state of the event process. The value is displayed in binary format.
- PREV_TIMESTAMP_TO_CHECK. A timestamp that the IEP Service Engine uses to maintain execution state for the event process.
- PREV_TIMESTAMP_TO_PROCESS. A timestamp that the IEP Service Engine uses to maintain execution state for the event process.

EMS_TABLE_USAGE_*N* Tables

The EMS_TABLE_USAGE_*N* tables are used in garbage collection. The EMS_TABLE_USAGE_*N* tables contain the following columns:

- TABLE_NAME. The name of a table.
- USER_ID. The ID of the operator that is using the table.
- EMS_TIMESTAMP. A timestamp that is used in garbage collection.

Operator-Specific Database Tables

In the IEP Service Engine, one or more tables are created per operator. These tables are specific to operator behavior and functionality.

Caution – Do not delete or alter these tables.

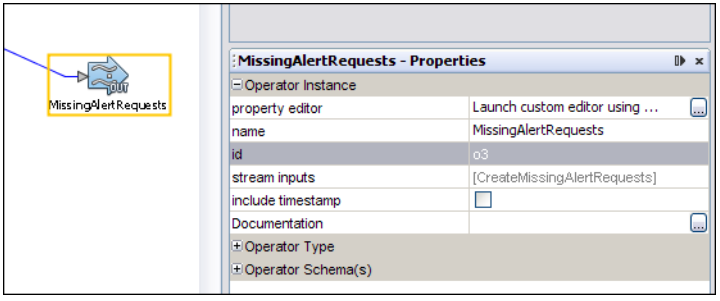
The following table names illustrate how the names are formatted.

- Q_0_O0
- Q_0_O1
- Q_1_O0
- Q_1_O1

The first character of the table name is always `Q`.

The character after the first underscore is the ID of the event processor in the EMS_PLAN table.

The characters after the second underscore are the ID of the operator. You can view the operator ID in the IEP design view by selecting the operator and displaying the Properties window.



The operator ID is automatically generated. You cannot change the ID.

Configuring Message Reliability in an IEP Module Project

The IEP Service Engine receives messages from and sends messages to other Java Business Integration (JBI) components. You can configure an IEP Module project for reliability in these message exchanges. If the system crashes and is subsequently restarted, message loss and message duplication do not occur.

This feature is supported for any JBI component that supports XA transactions.

You must follow the JBI component's procedure for configuring XA transactions.

Assume that you create an IEP Module project that receives messages from the JMS Binding Component and sends messages to the JMS Binding Component. To configure message reliability, set the `transaction` attribute of the JMS `operation` element to `XATransaction`.

The following example shows an input binding in a WSDL document. The `transaction` attribute is set to `XATransaction`.

```
<binding name="InputBinding"
         type="defns:InputPt">
    <jms:binding></jms:binding>
    <operation name="StreamInput0">
        <jms:operation destination="ext_client_publish_IEPCrashAndRecoveryNoBP"
                       destinationType="Queue"
                       transaction="XATransaction"
                       maxConcurrentConsumers="32">
        </jms:operation>
        <input>
            <jms:message textPart="input"
                         messageType="TextMessage">
            </jms:message>
        </input>
    </operation>
</binding>
```

The following example shows an output binding in a WSDL document. The `transaction` attribute is set to `XATransaction`.

```
<binding name="OutputBinding_StreamOutput0"
         type="defns:OutputPt_StreamOutput0">
    <jms:binding></jms:binding>
    <operation name="StreamOutput0">
        <jms:operation destination="ext_client_consume_IEPCrashAndRecoveryNoBP"
                       destinationType="Queue"
                       transaction="XATransaction"
                       deliveryMode="PERSISTENT"
                       disableMessageID="true"
                       disableMessageTimeStamp="true">
```

```
        </jms:operation>
        <input>
            <jms:message textPart="output"
                         messageType="TextMessage">
            </jms:message>
        </input>
    </operation>
</binding>
```

For more information about the JMS Binding Component, see *Using the JMS Binding Component*.

You can disable message reliability for outbound messages by editing the Transacted Output property of the IEP Service Engine.

▼ To Disable Message Reliability for Outbound Messages

1. **Go to the NetBeans IDE.**
2. **In the Services window, right-click the `sun-iep-engine` node and choose Properties.**
3. **Clear the Transacted Output check box.**
4. **Click OK.**

Index