SAP HANA Smart Data Streaming: Building Custom Adapters
## Content

1 **Introduction.** ................................................................. 4
   1.1 Input and Output Adapters. ........................................... 5
   1.2 Managed and Unmanaged Adapters. .............................. 6
   1.3 Subscribing to Data with Input Adapters. .................... 6
   1.4 Publishing Data with Output Adapters. ....................... 7
   1.5 Adapter Run States. .................................................. 8
   1.6 Editing Adapter Property Sets. ................................... 8
   1.7 Directories for Custom Adapters. ................................. 9
   1.8 Running Custom Adapters on a Remote Server. ............... 10
   1.9 Validating External Adapter Connectivity. .................. 11

2 **Smart Data Streaming Adapter Toolkit.** .......................... 13
   2.1 Preconfigured Adapters Included with the Adapter Toolkit. 15
   2.2 Create a Custom Adapter. .......................................... 18
   2.3 Accessing Adapter Toolkit API Reference Information. .... 19
   2.4 Transporter Modules. ................................................ 20
      Transporters Currently Available from SAP. .................. 21
      Building a Custom Transporter Module. ....................... 57
   2.5 Formatter Modules. .................................................. 59
      Formatters Currently Available from SAP. .................... 60
      Datatype Mapping for Formatters. ............................. 97
      Building a Custom Formatter Module. ......................... 98
      Datatype Mapping for JMS Map Formatter. .................... 101
   2.6 Batch Processing. .................................................... 103
   2.7 Working with Schemas. ............................................. 104
      Implementing Schema Discovery in a Custom Adapter. .... 105
   2.8 Guaranteed Delivery. ............................................... 105
      Enabling Guaranteed Delivery for an Input Transporter. ... 106
   2.9 EspConnector Modules. ............................................. 108
      EspSubscriber Configuration. .................................. 109
      EspMultiStreamSubscriber Configuration. .................... 111
      EspPublisher Configuration. .................................... 112
      EspMultiStreamPublisher Configuration. ..................... 114
   2.10 Smart Data Streaming Properties. ............................... 116
   2.11 Configuring a New Adapter. ..................................... 118
      Encrypting Adapter Configuration Files. ....................... 127
      Adapter Logging Configuration. ................................ 129
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.12 Starting an Adapter</td>
<td>131</td>
</tr>
<tr>
<td>2.13 Stopping an Adapter</td>
<td>133</td>
</tr>
<tr>
<td>2.14 Adapter Toolkit Examples</td>
<td>135</td>
</tr>
<tr>
<td>Running an Adapter Example</td>
<td>136</td>
</tr>
<tr>
<td>Running the Schema Discovery Adapter Example</td>
<td>137</td>
</tr>
<tr>
<td>2.15 Adapter Toolkit: Sample cnxml File for Input Adapters</td>
<td>138</td>
</tr>
<tr>
<td>2.16 Adapter Toolkit: Sample cnxml File for Output Adapters</td>
<td>139</td>
</tr>
<tr>
<td>2.17 Debugging a Custom Adapter</td>
<td>140</td>
</tr>
<tr>
<td>2.18 Statistics</td>
<td>143</td>
</tr>
<tr>
<td>3 Creating Custom External Adapters Using SDKs</td>
<td>144</td>
</tr>
<tr>
<td>3.1 Java External Adapters</td>
<td>144</td>
</tr>
<tr>
<td>Connecting to a Project</td>
<td>145</td>
</tr>
<tr>
<td>Creating a Publisher</td>
<td>145</td>
</tr>
<tr>
<td>Sample Java Code for addRow</td>
<td>146</td>
</tr>
<tr>
<td>Subscribing Using Callback</td>
<td>146</td>
</tr>
<tr>
<td>Subscribe Using Direct Access Mode</td>
<td>148</td>
</tr>
<tr>
<td>Publish Using Callback</td>
<td>148</td>
</tr>
<tr>
<td>3.2 C/C++ External Adapters</td>
<td>149</td>
</tr>
<tr>
<td>Getting a Project</td>
<td>149</td>
</tr>
<tr>
<td>Publishing and Subscribing</td>
<td>150</td>
</tr>
<tr>
<td>handleData</td>
<td>151</td>
</tr>
<tr>
<td>3.3 .NET External Adapters</td>
<td>152</td>
</tr>
<tr>
<td>Connecting to the Smart Data Streaming Server</td>
<td>152</td>
</tr>
<tr>
<td>Connecting to a Project</td>
<td>153</td>
</tr>
<tr>
<td>Publishing</td>
<td>153</td>
</tr>
<tr>
<td>Connecting to a Subscriber</td>
<td>154</td>
</tr>
<tr>
<td>Subscribing Using Callback Mode</td>
<td>154</td>
</tr>
<tr>
<td>4 Adapter Integration Framework</td>
<td>157</td>
</tr>
<tr>
<td>4.1 cnxml Configuration File</td>
<td>157</td>
</tr>
<tr>
<td>4.2 Creating a cnxml File for a Custom Adapter</td>
<td>158</td>
</tr>
<tr>
<td>Example cnxml Configuration File</td>
<td>160</td>
</tr>
<tr>
<td>External Adapter Properties</td>
<td>162</td>
</tr>
<tr>
<td>External Adapter Commands</td>
<td>163</td>
</tr>
<tr>
<td>User-Defined Parameters and Parameter Substitution</td>
<td>165</td>
</tr>
<tr>
<td>Autogenerated Parameter Files</td>
<td>166</td>
</tr>
<tr>
<td>configFilename Parameter</td>
<td>167</td>
</tr>
<tr>
<td>5 Adapter Parameter Datatypes</td>
<td>168</td>
</tr>
<tr>
<td>6 Methods of Defining Adapter Parameters for Managed and Unmanaged Mode</td>
<td>170</td>
</tr>
</tbody>
</table>
1 Introduction

SAP HANA smart data streaming includes an extensive set of input and output adapters that you can use to subscribe to and publish data. Additional specialized adapters for smart data streaming are available from SAP as optional add-ons. You can also write your own adapters to integrate into the smart data streaming server, and design them to handle various external requirements that standard adapters cannot.

SAP HANA smart data streaming provides an adapter toolkit for building custom external (Java) adapters. The benefits to the toolkit include:

- A quick, flexible method for creating custom adapters
- Standard modules that you can reuse to create your own custom adapters

SAP HANA smart data streaming also provides various SDKs that you can use to build custom external adapters in a number of programming languages, such as C, C++, Java, and .NET (C#, Visual Basic, and so on).

In this section:

- Input and Output Adapters [page 5]
  Input and output adapters enable SAP HANA smart data streaming to send and receive messages from dynamic and static external sources and destinations.

- Managed and Unmanaged Adapters [page 6]
  An external adapter can be developed and configured to support running in managed-only mode, unmanaged-only mode, or to support both modes.

- Subscribing to Data with Input Adapters [page 6]
  Subscribe to data from an external data source, and use an input stream or window to send the data to smart data streaming.

- Publishing Data with Output Adapters [page 7]
  Use an output stream or window to publish data from smart data streaming to an external data source.

- Adapter Run States [page 8]
  Adapters progress through a set of run states (RS) as they interact with smart data streaming.

- Editing Adapter Property Sets [page 8]
  Adapter property sets are reusable groups of properties that are stored in the project configuration file. Use the CCR Project Configuration editor in the smart data streaming plugin for SAP HANA studio to define adapter property sets and store them in the associated file.

- Directories for Custom Adapters [page 9]
  Place custom adapter files in directories relative to the STREAMING_CUSTOM_ADAPTERS_HOME environment variable.

- Running Custom Adapters on a Remote Server [page 10]
  Set up the components required to run a custom toolkit adapter on a remote server.

- Validating External Adapter Connectivity [page 11]
Ensure that the hostname of the server to which you are trying to connect can be resolved so that external adapters can successfully connect to these sources. You may be trying to connect to the smart data streaming server or other external sources.

### 1.1 Input and Output Adapters

Input and output adapters enable SAP HANA smart data streaming to send and receive messages from dynamic and static external sources and destinations.

External sources or destinations can include:
- Data feeds
- Sensor devices
- Messaging systems
- Radio frequency identification (RFID) readers
- Email servers
- Relational databases

Input adapters connect to an external data source and translate incoming messages from the external sources into a format that is accepted by the smart data streaming server. Output adapters translate rows that are published by smart data streaming into message formats that are compatible with external destinations and send those messages downstream.

The following illustration shows a series of input adapters that translate messages from a temperature sensor, bar code scanner, and a Java Message Service (JMS) cloud into formats that are compatible with smart data streaming. After the data is processed using various queries within smart data streaming, output adapters convert the resulting rows into updates that are sent to an external database server, email server, and Web services dashboard.

Figure 1: Overview of Input and Output Adapters in Use
1.2 Managed and Unmanaged Adapters

An external adapter can be developed and configured to support running in managed-only mode, unmanaged-only mode, or to support both modes.

Managed external adapters:

- Are started and stopped by the smart data streaming server with smart data streaming projects
- Have a `.cnxml` adapter configuration file that is configured within studio
- Are referenced in an ATTACH ADAPTER statement

Unmanaged external adapters:

- Start and stop independently of the smart data streaming server and smart data streaming projects
- Are configured independently
- Are not referenced in an ATTACH ADAPTER statement

1.3 Subscribing to Data with Input Adapters

Subscribe to data from an external data source, and use an input stream or window to send the data to smart data streaming.

Procedure

1. Assess the data to determine which sets or subsets of data you want to pull into smart data streaming.
2. Choose an input adapter that is suited for this task.
   - If the data source uses datatypes that are not supported by the smart data streaming server, the server maps the data to an accepted datatype. Review the associated mapping description for your adapter in the `SAP HANA Smart Data Streaming: Adapters Guide`.
3. Create an input stream or window.
4. Use the CREATE SCHEMA statement to define the structure for incoming data within this stream or window.
5. (Skip this step if using an unmanaged adapter) Use the ATTACH ADAPTER statement to attach your adapter to the newly created stream or window, and specify values for the adapter properties.
   - Use the DECLARE block and `parameters` qualifier to define default adapter parameter values before you attach your adapter. Once you create the ATTACH ADAPTER statement, you can set the adapter properties to the parameter values you declared.

   Note
   
   You can bind declared parameters to a new value only when a module or project is loaded.

6. Start the smart data streaming project. If you are using an unmanaged adapter, start it manually.
Next Steps

For detailed information about configuring individual smart data streaming-supplied adapters, datatype mappings, and schema discovery, see the SAP HANA Smart Data Streaming: Adapters Guide.

Backslashes in CCL files generate errors, because in that context, they are control characters. For detailed information on CCL queries and statements, such as the ATTACH ADAPTER, CREATE SCHEMA, and DECLARE statements, see the SAP HANA Smart Data Streaming: CCL Reference guide.

1.4 Publishing Data with Output Adapters

Use an output stream or window to publish data from smart data streaming to an external data source.

Procedure

1. Assess the output data to determine which sets or subsets of data you want to send to an external data source.
2. Choose an output adapter suited for this task.
   If the output destination uses datatypes that are not supported by the smart data streaming server, the server maps the data to an accepted datatype. Review the associated mapping description for your adapter in the SAP HANA Smart Data Streaming: Adapters Guide to ensure that the resulting datatype is permitted by the external data destination.
3. Create an output stream or window.
4. Use the CREATE SCHEMA statement to define the structure for outgoing data within this stream or window.
5. (Skip this step if using an unmanaged adapter) Use the ATTACH ADAPTER statement to attach your adapter to the output stream or window, and set values for the adapter properties.
   To declare default parameters for your adapter properties, use the DECLARE block and parameter qualifier to define default parameter values before you attach your adapter. Once you create the ATTACH ADAPTER statement, you can set the adapter properties to the parameter values you declared.

   Note
   You can bind declared parameters to a new value only when a module or project is loaded.
6. Start the smart data streaming project. If you are using an unmanaged adapter, start the adapter manually.
Next Steps

For detailed information on configuring individual smart data streaming-supplied adapters, datatype mappings, and schema discovery, see the *SAP HANA Smart Data Streaming: Adapters Guide*.

Backslashes in CCL files generate errors, because in that context, they are control characters. For detailed information on CCL queries and statements, such as the ATTACH ADAPTER, CREATE SCHEMA, and DECLARE statements, see the *SAP HANA Smart Data Streaming: CCL Reference guide*.

1.5 Adapter Run States

Adapters progress through a set of run states (RS) as they interact with smart data streaming.

- **RS_READY** – indicates that the adapter is ready to be started.
- **RS_INITIAL** – indicates that the adapter is performing start-up and initial loading.
- **RS_CONTINUOUS** – indicates that the adapter is continuously waiting for additional data.
- **RS_IDLE** – indicates that the adapter has timed out or is attempting to restore a broken socket.
- **RS_DONE** – indicates when the adapter no longer returns data and can no longer retrieve data following the poll period.
- **RS_DEAD** – indicates that the adapter has entered the exited state.

When polling is enabled, an input adapter may change states between RS_CONTINUOUS and RS_IDLE.

1.6 Editing Adapter Property Sets

Adapter property sets are reusable groups of properties that are stored in the project configuration file. Use the CCR Project Configuration editor in the smart data streaming plugin for SAP HANA studio to define adapter property sets and store them in the associated file.

Context

Property sets appear in a tree format, and individual property definitions are shown as children to property sets.

Procedure

1. From the SAP HANA Streaming Development perspective, double-click the .ccr file to open the CCR Project Configuration editor.
2. Select the Adapter Properties tab.

3. (Optional) To create a list of adapter property sets that correspond to the ATTACH ADAPTER statements in the main CCL file for the project, click Add from CCL.

4. To create a new adapter property set, click Add.

5. In the Property Set Details pane, define a name for the property set.

6. To add a new property to a property set, right-click the set and select New Property.

   **Note**
   
   You can add as many property items to a property set as required.

7. To configure a property:
   a. In the Property Details pane, define a name for the property.
   b. To define a property as an environment variable:
      - For Windows, use the format `%<environment-variable-name>%`.
      - For Unix, use the format `${<environment-variable-name>}`.
   c. Enter a value for the property.

8. (Optional) To encrypt the property value:
   a. Select the property value and click Encrypt.
   b. Enter the required fields, including Cluster URI and credential fields.
   c. Click Encrypt.
      
      The value, and related fields, are filled with randomized encryption characters.
   **Note**
   
   To reset the encryption, click Encrypt beside the appropriate field. Change the values, as appropriate, then click Reset.

9. To remove items from the All Adapter Properties list:
   - Right-click a property set and select Remove, or
   - Right-click a property and select Delete.

### 1.7 Directories for Custom Adapters

Place custom adapter files in directories relative to the STREAMING_CUSTOM_ADAPTERS_HOME environment variable.

### Systems with a Single Tenant

SAP HANA smart data streaming sets an environment variable, STREAMING_CUSTOM_ADAPTERS_HOME, for custom adapter files. This variable is initially set to `/usr/sap/<SID>/HDB<instance>/streaming/cluster/<sid>/adapters`. Custom adapters you place in this directory are available to all projects in your system.
Systems with Multiple Tenants

To make a custom adapter available only to an individual tenant database, set STREAMING_CUSTOM_ADAPTERS_HOME for that tenant’s environment to `/usr/sap/<SID>/HDB<instance>/streaming/cluster/<database-name>/adapters` and place files relative to this directory.

To make a custom adapter available to all tenant databases, temporarily set STREAMING_CUSTOM_ADAPTERS_HOME to `/usr/sap/<SID>/HDB<instance>/streaming/STREAMING-2_0/adapters/framework` and place files relative to this directory. Files in this location may be overwritten when upgrading the SAP HANA system.

**Caution**

In a low-isolation multitenant system, do not set STREAMING_CUSTOM_ADAPTERS_HOME by default, as other tenants use separate folders and setting the variable incorrectly can cause errors.

**Note**

The STREAMING_CUSTOM_ADAPTERS_HOME environment variable must be set correctly for command-line utilities to function.

Directory Structure

Place custom adapter files in the following locations:

<table>
<thead>
<tr>
<th>File</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>custommodulesdefine.xml, parametersdefine.xsd</td>
<td>STREAMING_CUSTOM_ADAPTERS_HOME/config</td>
</tr>
<tr>
<td>Adapter libraries</td>
<td>STREAMING_CUSTOM_ADAPTERS_HOME/libj</td>
</tr>
<tr>
<td>.cnxml files</td>
<td>STREAMING_CUSTOM_ADAPTERS_HOME/cnxml</td>
</tr>
</tbody>
</table>

1.8 Running Custom Adapters on a Remote Server

Set up the components required to run a custom toolkit adapter on a remote server.

**Context**

To run custom adapters developed using the adapter toolkit on a remote server, first set up adapter toolkit directories on the target machine.
**Procedure**

1. On the remote server, create a new folder to house your adapter toolkit directory.
2. Set the STREAMING_HOME environment variable to point to the new folder.
   a. Go to Control Panel, then select [System ➤ Advanced System Settings ➤ Environment Variables](#).
   b. Set the STREAMING_HOME variable to the newly created adapter directory.
3. From an existing installation of SAP HANA smart data streaming, copy the following directories into the new $STREAMING_HOME location:
   - $STREAMING_HOME/adapters/framework
   - $STREAMING_HOME/lib/jre
   Ensure that the original directory structure under STREAMING_HOME is preserved on the remote server.

   **Note**
   If you prefer to use your own copy of the JRE, use version 6 or higher. Modify the scripts under $STREAMING_HOME/adapters/framework/bin to point to the correct JRE location.

4. (Optional) If you are running only custom adapters on the remote server, and none of the preconfigured adapters included with smart data streaming, remove:
   - $STREAMING_HOME/adapters/framework/examples
   - $STREAMING_HOME/adapters/framework/instances

**1.9 Validating External Adapter Connectivity**

Ensure that the hostname of the server to which you are trying to connect can be resolved so that external adapters can successfully connect to these sources. You may be trying to connect to the smart data streaming server or other external sources.

**Context**

Use the `ping` command to verify that the hostname can be resolved. The following is an example error message that occurs when the server is not being resolved:

```
Failed call to:https://<hostname>:61308/RPC2 (Failed to read server's response: <hostname>) java.io.IOException: Failed call to:https://<hostname>:61308/RPC2 (Failed to read server's response: <hostname>)
```
Use the following procedure if you cannot resolve the hostname:

**Procedure**

1. Determine the IP address of the host on which the server is running. For example, to determine the IP address of the smart data streaming server host, run this command from that machine:

   ```
   nslookup <hostname>
   ```

2. Add the following line to `C:\Windows\System32\drivers\etc\hosts` (Windows) or `/etc/hosts` (UNIX):

   ```
   <ip-address-of-server-hostname> <server-hostname>
   ```
Use the smart data streaming adapter toolkit to quickly build custom external adapters using Java. Adapters built using the toolkit consist of various component modules configured together to deliver data to and publish data from smart data streaming. Module types include transporters (for interacting with external transports), formatters (for converting data from one format to another), and smart data streaming connectors (for subscribing or publishing to smart data streaming).

The toolkit includes numerous transporters, formatters, and smart data streaming connectors that can be configured in various combinations by an administrator. You can also combine these out-of-the-box modules with custom modules created by a Java developer.

The adapter toolkit allows you to implement:

- An input adapter to act as a data source for the smart data streaming server
- An output adapter to act as a data destination and deliver data from streams in smart data streaming
- A dual-direction adapter to act as both a data source and data destination for smart data streaming
- Guaranteed delivery (GD) to minimize loss of data during transfer of input data
- Schema discovery to automatically discover schemas for your custom input and output adapters
SAP HANA smart data streaming includes various preconfigured and ready-to-use adapters that have been created using the adapter toolkit.

Create a Custom Adapter

Use the smart data streaming adapter toolkit to create a custom adapter. You can do this by combining transporter and formatter modules that are provided with the adapter toolkit, by writing your own custom transporter and formatter modules, or by combining existing modules with custom ones.

Accessing Adapter Toolkit API Reference Information

API documentation contains information about methods, functions, and other programming building blocks.

Transporter Modules

A transporter module is the interface that interacts with external data sources by obtaining data from a data source or outputting data to a data destination.

Formatter Modules

A formatter module converts between the data format of the transporter module and smart data streaming.

Batch Processing

Details about how to control the manner in which AdapterRow instances are sent and processed by smart data streaming.

Working with Schemas

Discover external schemas and create CCL schemas, streams, or windows based on the format of the data from the data source that is connected to an adapter.

Guaranteed Delivery

Guaranteed delivery (GD) is a delivery mechanism used to guarantee that data is processed from a stream or window to an adapter.

EspConnector Modules

The EspConnector modules are responsible for connecting to smart data streaming. Connector module types include EspSubscriber, EspMultiStreamSubscriber, EspPublisher, and EspMultiStreamPublisher.

Smart Data Streaming Properties

Edit the smart data streaming elements in the adapter configuration (.xml) file to connect a project to an adapter instance.

Configuring a New Adapter

Configure a new adapter by creating a configuration file for it. The configuration file defines the adapter component chain through which data is processed, and the connection to smart data streaming.

Starting an Adapter

Start the adapter either in unmanaged, managed, or cluster-managed mode. In unmanaged and cluster-managed mode, the adapter is started separately from the smart data streaming project, and in managed mode, the adapter is started with the smart data streaming project.

Stopping an Adapter
Stop the adapter either in unmanaged, managed, or cluster-managed mode. In unmanaged and cluster-managed mode, the adapter is stopped separately from the smart data streaming project. In managed mode, the adapter is stopped with the smart data streaming project.

**Adapter Toolkit Examples [page 135]**

The `%STREAMING_HOME%/adapters/framework/examples` directory contains various example modules for the adapter toolkit.

**Adapter Toolkit: Sample cnxml File for Input Adapters [page 138]**

Adapter type: toolkit_input. Sample `cnxml` file for an input adapter created using the adapter toolkit. You can use this file as reference for creating your own `cnxml` file for your custom adapter.

**Adapter Toolkit: Sample cnxml File for Output Adapters [page 139]**

Adapter type: toolkit_output. Sample `cnxml` file for an output adapter created using the adapter toolkit. You can use this file as reference for creating your own `cnxml` file for your custom adapter.

**Debugging a Custom Adapter [page 140]**

Debug a custom Java adapter (that was built using the adapter toolkit) by starting it in debug mode and using an Integrated Development Environment (IDE) that supports remote debugging, such as Eclipse.

**Statistics [page 143]**

An adapter constructed using the SAP HANA smart data streaming adapter toolkit maintains a set of common statistics to show its status and track its loading activities.

### 2.1 Preconfigured Adapters Included with the Adapter Toolkit

SAP HANA smart data streaming includes various preconfigured and ready-to-use adapters that have been created using the adapter toolkit.

You can use these adapters as reference examples when creating your own adapters. You can also reuse individual transporter and formatter modules from these adapters in your own custom adapters.

For detailed information about these adapters, see the *SAP HANA Smart Data Streaming: Adapters Guide*.

**Input Adapters**

<table>
<thead>
<tr>
<th>Adapter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File/Hadoop CSV Input</td>
<td>Obtains CSV data from files on a local hard disk and publishes it to smart data streaming.</td>
</tr>
<tr>
<td>File/Hadoop Event XML Input</td>
<td>Reads XML list text files and inputs this data into smart data streaming.</td>
</tr>
<tr>
<td>Adapter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>File/Hadoop JSON Input</td>
<td>Takes JSON messages from JSON files, and publishes them to smart data streaming.</td>
</tr>
<tr>
<td>File/Hadoop XML Input</td>
<td>Loads data from an XML document into a project in smart data streaming.</td>
</tr>
<tr>
<td>FTP CSV Input</td>
<td>Obtains CSV data from an FTP server and publishes it to smart data streaming.</td>
</tr>
<tr>
<td>FTP Event XML Input</td>
<td>Reads data from an XML document on an FTP server into smart data streaming.</td>
</tr>
<tr>
<td>JDBC Input</td>
<td>Receives data from tables in a database and inputs it into smart data streaming.</td>
</tr>
<tr>
<td>JMS CSV Input</td>
<td>Reads CSV data from a JMS server and outputs this data into smart data streaming.</td>
</tr>
<tr>
<td>JMS Object Input</td>
<td>Receives object array data from a JMS server and publishes it to smart data streaming.</td>
</tr>
<tr>
<td>JMS Event XML Input</td>
<td>Obtains XML list string messages from a JMS server and publishes them to smart data streaming.</td>
</tr>
<tr>
<td>Kafka Avro Record Input</td>
<td>Obtains data in Avro record binary format from a Kafka server and publishes it to smart data streaming.</td>
</tr>
<tr>
<td>Kafka CSV Input</td>
<td>Obtains CSV data from the Kafka server and publishes it to smart data streaming.</td>
</tr>
<tr>
<td>Kafka Event XML Input</td>
<td>Obtains event XML data from a Kafka server and publishes it to smart data streaming.</td>
</tr>
<tr>
<td>Kafka JSON Input</td>
<td>Obtains JSON data from a Kafka server and publishes it to smart data streaming.</td>
</tr>
<tr>
<td>Kafka String Input</td>
<td>Obtains string data from a Kafka server and publishes it to smart data streaming.</td>
</tr>
<tr>
<td>SAP RFC Input</td>
<td>Executes RFCs to import data from SAP systems into smart data streaming.</td>
</tr>
<tr>
<td>Socket CSV Input</td>
<td>Obtains CSV string data from a Socket server and publishes it to smart data streaming.</td>
</tr>
<tr>
<td>Socket JSON Input</td>
<td>Obtains streaming data from the socket server, formats data into JSON format and inputs it into smart data streaming.</td>
</tr>
</tbody>
</table>
### Adapter Name

<table>
<thead>
<tr>
<th>Adapter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket Event XML Input</td>
<td>Obtains XML data from a Socket server and publishes it to smart data streaming.</td>
</tr>
<tr>
<td>Web Services (SOAP) Input</td>
<td>Connects to a Web service to obtain data to feed into smart data streaming.</td>
</tr>
</tbody>
</table>

### Output Adapters

<table>
<thead>
<tr>
<th>Adapter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File/Hadoop CSV Output</td>
<td>Reads rows from smart data streaming and writes this data into a specified CSV file.</td>
</tr>
<tr>
<td>File/Hadoop Event XML Output</td>
<td>Reads rows from smart data streaming and writes this data into XML list files.</td>
</tr>
<tr>
<td>File/Hadoop JSON Output</td>
<td>Takes data from smart data streaming, formats it into JSON format, and sends it to a JSON file.</td>
</tr>
<tr>
<td>File/Hadoop XML Output</td>
<td>Outputs data from a project in smart data streaming into an XML document.</td>
</tr>
<tr>
<td>FTP CSV Output</td>
<td>Takes data from smart data streaming, formats it to CSV format, and saves it to a file on an FTP server.</td>
</tr>
<tr>
<td>FTP Event XML Output</td>
<td>Reads XML data from a project for smart data streaming, writes it to an XML document, and uploads this file to the FTP server.</td>
</tr>
<tr>
<td>HTTP Client JSON Output</td>
<td>Sends JSON-formatted data to an HTTP server using an OData POST request.</td>
</tr>
<tr>
<td>HTTP Client Output</td>
<td>Obtains stream data from smart data streaming and outputs it to an HTTP server.</td>
</tr>
<tr>
<td>HTTP Client XML Output</td>
<td>Sends XML-formatted data to an HTTP server using an OData POST request.</td>
</tr>
<tr>
<td>JDBC Output</td>
<td>Sends data from smart data streaming to a database table.</td>
</tr>
<tr>
<td>JMS CSV Output</td>
<td>Sends CSV data from smart data streaming to a JMS server.</td>
</tr>
<tr>
<td>JMS Object Output</td>
<td>Takes data from smart data streaming, formats it into object array format, and sends it to a JMS server.</td>
</tr>
<tr>
<td>JMS Event XML Output</td>
<td>Takes XML data from smart data streaming, formats it to XML list format, and sends it to a JMS server.</td>
</tr>
<tr>
<td>Adapter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Kafka Avro Record Output</td>
<td>Reads data from smart data streaming, formats it to Avro format, and writes it to a Kafka server.</td>
</tr>
<tr>
<td>Kafka CSV Output</td>
<td>Reads data from smart data streaming, transforms it to CSV format, and writes it to a Kafka server.</td>
</tr>
<tr>
<td>Kafka Event XML Output</td>
<td>Reads data from smart data streaming, formats it to event XML format, and writes it to a Kafka server.</td>
</tr>
<tr>
<td>Kafka JSON Output</td>
<td>Reads data from smart data streaming, formats it to JSON format, and writes it to a Kafka server.</td>
</tr>
<tr>
<td>Kafka String Output</td>
<td>Reads data from smart data streaming, formats it to string format, and writes it to a Kafka server.</td>
</tr>
<tr>
<td>SAP RFC Output</td>
<td>Exports data from smart data streaming into SAP systems.</td>
</tr>
<tr>
<td>Socket CSV Output</td>
<td>Takes data from smart data streaming, formats it into CSV format, and outputs it to a Socket server.</td>
</tr>
<tr>
<td>Socket JSON Output</td>
<td>Takes JSON data from smart data streaming, formats it to ByteBuffer, and transports it to the Socket server in streaming format.</td>
</tr>
<tr>
<td>Socket Event XML Output</td>
<td>Takes data from smart data streaming, formats it to XML list format, and outputs it to a Socket server.</td>
</tr>
<tr>
<td>Web Services (SOAP) Output</td>
<td>Delivers output from smart data streaming to a Web service.</td>
</tr>
</tbody>
</table>

## 2.2 Create a Custom Adapter

Use the smart data streaming adapter toolkit to create a custom adapter. You can do this by combining transporter and formatter modules that are provided with the adapter toolkit, by writing your own custom transporter and formatter modules, or by combining existing modules with custom ones.

1. **Building a Custom Transporter Module [page 57]**
   Use the smart data streaming adapter toolkit to build a custom transporter module to use within the adapter instance of your choice.

2. **Building a Custom Formatter Module [page 98]**
   Use the smart data streaming adapter toolkit to build a custom formatter module to use within the adapter instance of your choice.

3. **Enabling Guaranteed Delivery for an Input Transporter [page 106]**
   (Optional) Enable guaranteed delivery (GD) in a custom input transporter by implementing the com.sybase.esp.adapter.framework.event.AdapterRowEventListener interface, registering the GdAdapterEventListener class, and adding and setting the `<GDMode>` parameter to true for the EspPublisher or EspMultistreamPublisher.
4. **Implementing Schema Discovery in a Custom Adapter [page 105]**
   (Optional) Use interfaces and functions from the adapter toolkit to implement schema discovery in a transporter and formatter module. The two types of schema discovery are non-sampling and sampling. Use non-sampling schema discovery when the transporter can fully determine schema on its own. Use sampling schema discovery when the transporter cannot determine the schema and passes this data to the formatter to generate the schema.

5. **Configuring a New Adapter [page 118]**
   Configure a new adapter by creating a configuration file for it. The configuration file defines the adapter component chain through which data is processed, and the connection to smart data streaming.

6. **Creating a cnxml File for a Custom Adapter [page 158]**
   Create a .cnxml configuration file for your custom external adapter so that you can configure the adapter in the SAP HANA Streaming Development perspective, and start and stop it with a project for smart data streaming.

7. **Starting an Adapter [page 131]**
   Start the adapter either in unmanaged, managed, or cluster-managed mode. In unmanaged and cluster-managed mode, the adapter is started separately from the smart data streaming project, and in managed mode, the adapter is started with the smart data streaming project.

8. **Stopping an Adapter [page 133]**
   Stop the adapter either in unmanaged, managed, or cluster-managed mode. In unmanaged and cluster-managed mode, the adapter is stopped separately from the smart data streaming project. In managed mode, the adapter is stopped with the smart data streaming project.

**Related Information**

- Accessing Adapter Toolkit API Reference Information [page 19]
- Debugging a Custom Adapter [page 140]
- Formatter Modules [page 59]
- Transporter Modules [page 20]
- EspConnector Modules [page 108]

**2.3 Accessing Adapter Toolkit API Reference Information**

API documentation contains information about methods, functions, and other programming building blocks.

**Procedure**

1. **Navigate to** %STREAMING_HOME%\doc\adaptertoolkit on Windows or $STREAMING_HOME/doc/adaptertoolkit on Linux.
2. **Launch** index.html.
2.4 Transporter Modules

A transporter module is the interface that interacts with external data sources by obtaining data from a data source or outputting data to a data destination.

SAP HANA smart data streaming supports two types of transporters: row-based and stream-based.

Row-based transporters obtain and output data in row format, such as a database transporter. These transporters work with AdapterRow instances, which are containers for one or more records or rows as they flow from one module (transporter, formatter, or smart data streaming connector) to the next. You can add multiple records as objects within a List of a single AdapterRow object. The AdapterRow has a timestamp and block flags that control how its records are communicated to and from smart data streaming. See Event Blocks in the SAP HANA Smart Data Streaming: CCL Reference for additional details.

Stream-based transporters deal with streaming data, such as a socket transporter. These transporters work with ByteStream or ByteBuffer instances, which represent a continuous stream of data.

In this section:

- **Transporters Currently Available from SAP [page 21]**
  - The adapter toolkit includes numerous transporter modules with smart data streaming. You can reuse these modules to create a custom adapter instance.

- **Building a Custom Transporter Module [page 57]**
  - Use the smart data streaming adapter toolkit to build a custom transporter module to use within the adapter instance of your choice.

Related Information

- Format器 Modules [page 59]
- EspConnector Modules [page 108]
- Accessing Adapter Toolkit API Reference Information [page 19]
2.4.1 Transporters Currently Available from SAP

The adapter toolkit includes numerous transporter modules with smart data streaming. You can reuse these modules to create a custom adapter instance.

Input transporters obtain data from external data sources and send this data out into smart data streaming. The format of this output data is specified in the Output Datatype column of the following table.

AdapterRow is a container for one or more records or rows as they flow from one module (transporter, formatter, or the smart data streaming connector) to the next. You can add multiple records as objects within a List of a single AdapterRow object. The AdapterRow has a timestamp and block flags that control how its records are communicated to and from smart data streaming. See Event Blocks in the SAP HANA Smart Data Streaming: CCL Reference for additional details.

AepRecord is the class that represents stream records. This class is the type that the smart data streaming publisher expects as the data member in the AdapterRow instances that it receives from the previous module. This class is also the type used in the AdapterRow instances that the smart data streaming subscriber passes on to the next module. The default operation code (opcode) for AepRecord is INSERT. Change this opcode based on your needs.

The following table describes the current standard input transporters available from SAP and specifies the output datatype for each of them:

Table 2:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mode (Streaming/Row)</th>
<th>Output Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Input Transporter</td>
<td>Can be both (depends on configuration)</td>
<td>Java.lang.String or Java.nio.ByteBuffer</td>
<td>In row mode, the transporter reads data from local files, wraps data with string, and sends it to the next module that is configured in the adapter configuration file. In streaming mode, the transporter reads data from local files, wraps it with ByteStream, and passes it to the next module that is configured in the adapter configuration file.</td>
</tr>
<tr>
<td>Name</td>
<td>Mode (Streaming/Row)</td>
<td>Output Datatype</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------</td>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FTP Input Transporter</td>
<td>Streaming</td>
<td>Java.nio.ByteBuffer</td>
<td>Reads binary data from files on an FTP server, wraps it up with ByteStream, and passes it to the next module that is configured in the adapter configuration file.</td>
</tr>
<tr>
<td>JDBC Input Transporter</td>
<td>Row</td>
<td>Java.util.List&lt;Java.lang.Object&gt;</td>
<td>Reads database records from a database using JDBC, and sends data records to the next module that is configured in the adapter configuration file.</td>
</tr>
<tr>
<td>JMS Input Transporter</td>
<td>Row</td>
<td>Java.lang.String or AepRecord or Java.util.List&lt;Java.lang.Object&gt;</td>
<td>Receives JMS messages from a JMS server, and sends this data to the next module that is configured in the adapter configuration file.</td>
</tr>
<tr>
<td>Kafka Input Transporter</td>
<td>Row</td>
<td>Java.lang.String or AvroRecord</td>
<td>Receives string or Avro data from a Kafka server, and sends it to the next module configured in the adapter configuration file.</td>
</tr>
<tr>
<td>Socket Input Transporter</td>
<td>Streaming</td>
<td>Java.nio.ByteBuffer</td>
<td>Reads binary data from a socket interface, wraps it with ByteStream, and passes it to the next module that is configured in the adapter configuration file.</td>
</tr>
</tbody>
</table>
Output transporters obtain data from smart data streaming and send it out to external data sources. The following table describes the current standard output transporters available from SAP and specifies the input datatype for each of them:

<table>
<thead>
<tr>
<th>Name</th>
<th>Mode (Streaming/Row)</th>
<th>Input Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Output Transporter</td>
<td>Can be both (depends on configuration)</td>
<td>Java.lang.String or Java.nio.ByteBuffer</td>
<td>In row mode, the transporter obtains string data from the previous module configured in the adapter configuration file, and writes this data to a local file. In streaming mode, the transporter obtains ByteStream data from the previous module configured in the adapter configuration file, and writes this data to a local file.</td>
</tr>
<tr>
<td>FTP Output Transporter</td>
<td>Streaming</td>
<td>Java.nio.ByteBuffer</td>
<td>Obtains ByteStream data from the previous module that is configured in the adapter configuration file, and saves it to a file on an FTP server.</td>
</tr>
<tr>
<td>HTTP Output Transporter</td>
<td>Row</td>
<td>AepRecord</td>
<td>Obtains data from the smart data streaming stream and outputs it to an HTTP server.</td>
</tr>
<tr>
<td>JDBC Output Transporter</td>
<td>Row</td>
<td>Java.util.List&lt;Java.lang.Object&gt;</td>
<td>Obtains row-based data from the previous module that is configured in the adapter configuration file, and saves it into a database table using JDBC.</td>
</tr>
<tr>
<td>JMS Output Transporter</td>
<td>Row</td>
<td>Java.lang.String or AepRecord or Java.util.List&lt;Java.lang.Object&gt;</td>
<td>Obtains data from the previous module that is configured in the adapter configuration file and sends it to a JMS server.</td>
</tr>
<tr>
<td>Kafka Output Transporter</td>
<td>Row</td>
<td>Java.lang.String or AvroRecord</td>
<td>Obtains data from the previous module that is configured in the adapter configuration file and sends it to a Kafka server.</td>
</tr>
<tr>
<td>Name</td>
<td>Mode (Streaming/Row)</td>
<td>Input Datatype</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------</td>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Socket Output Transporter</td>
<td>Streaming</td>
<td>Java.nio.ByteBuffer</td>
<td>Obtains ByteStream data from the previous module that is configured in the adapter configuration file, and outputs it through a socket interface.</td>
</tr>
</tbody>
</table>

**In this section:**

**File Input Transporter Module Properties [page 25]**

The File Input transporter reads data from local files, wraps the data with `String`, and sends it to the next module specified in the adapter configuration file. Set values for this transporter in the adapter configuration file.

**File Output Transporter Module Properties [page 30]**

The File Output transporter obtains data from the previous module specified in the adapter configuration file and writes it to local files. Set values for this transporter in the adapter configuration file.

**FTP Input Transporter Module Properties [page 35]**

The FTP Input transporter reads binary data from files on an FTP server, wraps it up with `ByteBuffer`, and sends it to the next module that is configured in the adapter configuration file. Set values for this transporter in the adapter configuration file.

**FTP Output Transporter Module Properties [page 36]**

The FTP Output transporter obtains data from the previous module configured in the adapter configuration file, and saves it to files on the FTP server. Set values for this transporter in the adapter configuration file.

**HTTP Output Transporter Module Properties [page 37]**

The HTTP Output transporter obtains stream data from smart data streaming and sends it to an HTTP server. Set values for this transporter in the adapter configuration file.

**JDBC Input Transporter Module Properties [page 40]**

The JDBC Input transporter reads database records using JDBC and sends them to the next module specified in the adapter configuration file. Set values for this transporter in the adapter configuration file.

**JDBC Output Transporter Module Properties [page 41]**

The JDBC Output transporter obtains data from the previous module specified in the adapter configuration file and writes it into a database table using JDBC. Set values for this transporter in the adapter configuration file.

**JMS Input Transporter Module Properties [page 44]**

The JMS Input transporter receives JMS messages from a JMS server, and sends it to the next module that is configured in the adapter configuration file. Set values for this transporter in the adapter configuration file.

**JMS Output Transporter Module Properties [page 47]**

The JMS Output transport obtains data from the previous module that is configured in the adapter configuration file, wraps it up, and sends it to a JMS server. Set values for this transporter in the adapter configuration file.
Kafka Input Transporter Module Properties [page 50]
The Kafka Input transporter acts as a Kafka consumer. It reads data from a Kafka server and sends string data to smart data streaming.

Kafka Output Transporter Module Properties [page 53]
The Kafka Output transporter acts as a Kafka producer. It reads data from smart data streaming and writes string data asynchronously to a Kafka server.

Socket Input Transporter Module Properties [page 54]
The Socket Input transporter reads binary data from the socket interface, wraps it with ByteBuffer, and sends it to the next module that is configured in the adapter configuration file. Set values for this transporter in the adapter configuration file.

Socket Output Transporter Module Properties [page 55]
The Socket Output transporter obtains data from the previous module configured in the adapter configuration file, and outputs it using the socket interface. Set values for this transporter in the adapter configuration file.

Related Information

Accessing Adapter Toolkit API Reference Information [page 19]
Building a Custom Transporter Module [page 57]

2.4.1.1 File Input Transporter Module Properties

The File Input transporter reads data from local files, wraps the data with string, and sends it to the next module specified in the adapter configuration file. Set values for this transporter in the adapter configuration file.

The File Input transporter supports schema discovery.
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| Dir         | Type: string  
(Required) The path to the data file that you want the adapter to read. The path is relative to the adapter sandbox base directory /adapters/. In a multitenant database container, specify the path relative to the project or adapter working directory. For example, /hana/data_streaming/HDC/sds_tenant/adapters/<workspace>. No default value.  
To use Hadoop system files, use an HDFS folder URI instead of a local file system folder. For example, hdfs://<hdfs-server>:9000/<folder-name>/<sub-folder-name>/<leaf-folder-name>.  
For version 1.2.1, copy the hadoop-core-1.2.1.jar file to $STREAMING_CUSTOM_ADAPTERS_HOME/libj.  
For version 2.2.0, copy these files over to $STREAMING_CUSTOM_ADAPTERS_HOME/libj:  
- hadoop-common-2.2.0.jar  
- hadoop-auth-2.2.0.jar  
- hadoop-hdfs-2.2.0.jar  
- guava-11.0.2.jar  
- protobuf-java-2.5.0.jar  
For version 2.3 to 2.5, copy these files over to $STREAMING_CUSTOM_ADAPTERS_HOME/libj:  
- guava-11.0.2.jar  
- hadoop-auth-2.x.x.jar  
- hadoop-common-2.x.x.jar  
- hadoop-hdfs-2.x.x.jar  
- protobuf-java-2.5.0.jar  
For version 2.6, copy these files over to $STREAMING_CUSTOM_ADAPTERS_HOME/libj:  
- guava-11.0.2.jar  
- hadoop-auth-2.6.0.jar  
- hadoop-common-2.6.0.jar  
- hadoop-hdfs-2.6.0.jar  
- htrace-core-3.0.4.jar  
- protobuf-java-2.5.0.jar  
For version 2.7, copy these files over to $STREAMING_CUSTOM_ADAPTERS_HOME/libj:
### XML Element

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
</table>
| • commons-io-2.4.jar  
• guava-11.0.2.jar  
• hadoop-auth-2.7.0.jar  
• hadoop-common-2.7.0.jar  
• hadoop-hdfs-2.7.0.jar  
• htrace-core-3.1.0-incubating.jar  
• protobuf-java-2.5.0.jar  
Use a forward slash for both UNIX and Windows paths. |

### File

**Type:** string  
(Required) The relative path to the file you want the adapter to read or the regex pattern to filter the files on a given directory. See the DynamicMode element.  
If the Dir property is left blank, use this property to specify the absolute path to the data files that you want the adapter to read.  
No default value.

### AccessMode

**Type:** string  
(Required) Indicates an access mode:  
- **rowBased**  
The adapter reads one text line at a time.  
- **Streaming**  
The adapter reads a preconfigured size of bytes into a buffer.  
No default value.
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| DynamicMode     | Type: string  
(Advanced) Indicates a dynamic mode:  
**Static**  
The adapter reads the file specified in the Dir and File elements.  
**dynamicFile**  
The adapter reads the file specified in the Dir and File elements and keeps polling the new appended content. The polling period is specified in the PollingPeriod element.  
**dynamicPath**  
The adapter polls all new files under the Dir element. The File element acts as a regex pattern and filters out the necessary files.  
**dynamicFileAndPath**  
The adapter reads all files and polls for new files under the Dir element, and looks for new records in existing files. The File element acts as a regex pattern and filters out the necessary files.  
**rollingLogFiles**  
If PollingPeriod is 0, rollingLogFiles is automatically disabled. If PollingPeriod is greater than 0, during adapter startup, all content is read from the log file specified in the File element.  
When a log file rolls over, the adapter reads unread content from backup files and any new content since the last polling period.  
If DynamicMode has been set to dynamicPath and you leave the File element empty, the adapter reads all the files under the specified directory.  
An example regex pattern is ".*\.txt", which selects only files that end with ".txt". In regex patterns, you must include an escape character, \", before meta chars to include them in the pattern string.  
The default value is Static. |
| PollingPeriod   | Type: integer  
(Advanced) The period, in seconds, to poll the specified file or directory. Set this element only if the value of the DynamicMode element is set to dynamicFile, dynamicPath, dynamicFileAndPath, or rollingLogFiles.  
The default value is 0, which, along with all values less than 0, turns off polling. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RemoveAfterProcess</strong></td>
<td>Type: boolean</td>
</tr>
<tr>
<td></td>
<td>(Optional) If this property is set to true, the file is removed from the directory after the adapter processes it. This element takes effect if the value of the DynamicMode element is set to dynamicPath and ignored if it is set to dynamicFile instead. The default value is false.</td>
</tr>
<tr>
<td><strong>ScanDepth</strong></td>
<td>Type: integer</td>
</tr>
<tr>
<td></td>
<td>(Optional) The depth of the schema discovery. The adapter reads the number of rows specified by this element value when discovering the input data schema. The default value is 3.</td>
</tr>
</tbody>
</table>
2.4.1.2 File Output Transporter Module Properties

The File Output transporter obtains data from the previous module specified in the adapter configuration file and writes it to local files. Set values for this transporter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| Dir         | Type: string

(Required) The path to the data file that you want the adapter to write to. The path is relative to the adapter sandbox base directory /adapters/. In a multitenant database container, specify the path relative to the project or adapter working directory. For example, /hana/data_streaming/HDC/sds_tenant/adapters/<workspace>

The default value is ".", meaning the current directory in which the adapter is running. Alternatively, you can leave this value empty and specify the absolute path in the File property.

To use Hadoop system files, use an HDFS folder instead of a local file system folder. For example, hdfs://<hdfs-server>:9000/<folder-name>/<sub-folder-name>/<leaf-folder-name>


For version 1.2.1, copy the hadoop-core-1.2.1.jar file to $STREAMING_CUSTOM_ADAPTERS_HOME/libj.

For version 2.2.0, copy these files over to $STREAMING_CUSTOM_ADAPTERS_HOME/libj:

- hadoop-common-2.2.0.jar
- hadoop-auth-2.2.0.jar
- hadoop-hdfs-2.2.0.jar
- guava-11.0.2.jar
- protobuf-java-2.5.0.jar

For version 2.3 to 2.5, copy these files over to $STREAMING_CUSTOM_ADAPTERS_HOME/libj:

- guava-11.0.2.jar
- hadoop-auth-2.x.x.jar
- hadoop-common-2.x.x.jar
- hadoop-hdfs-2.x.x.jar
- protobuf-java-2.5.0.jar

For version 2.6, copy these files over to $STREAMING_CUSTOM_ADAPTERS_HOME/libj:

- guava-11.0.2.jar
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
|             | • hadoop-auth-2.6.0.jar  
|             | • hadoop-common-2.6.0.jar  
|             | • hadoop-hdfs-2.6.0.jar  
|             | • htrace-core-3.0.4.jar  
|             | • protobuf-java-2.5.0.jar  
|             | For version 2.7, copy these files over to $STREAMING_CUSTOM_ADAPTERS_HOME/libj:  
|             | • commons-io-2.4.jar  
|             | • guava-11.0.2.jar  
|             | • hadoop-auth-2.7.0.jar  
|             | • hadoop-common-2.7.0.jar  
|             | • hadoop-hdfs-2.7.0.jar  
|             | • htrace-core-3.1.0-incubating.jar  
|             | • protobuf-java-2.5.0.jar  
|             | Use a forward slash for both UNIX and Windows paths.  

File Type: string  
(Required) The relative path to the file to which the adapter writes. If the Dir property is left blank, use this property to specify the absolute path to the data file to which you want the adapter to write. No default value.

AccessMode Type: string  
(Required) Indicates an access mode:  
**rowBased**  
The adapter writes one text line at a time into the file.  
**Streaming**  
The adapter writes the raw data in ByteBuffer into the file. No default value.
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| AppendMode  | Type: boolean  
(Advanced) Indicates whether data is appended to an existing file. 
If set to true, and if there is an existing output file (<file-name>.csv, where <file-name> is derived from the File property), the adapter appends the data into the output file. 
If set to false, the adapter overwrites the existing file.  
**Note**  
AppendMode only affects the behavior of the output file if the file exists when the adapter starts. 
If the adapter is running in GD mode, set this property to true. Otherwise, the content of the output file gets erased every time the adapter is restarted. 
The default value is false. |
| BatchSize   | Type: integer  
(Advanced) If the adapter is running in GD (guaranteed delivery) mode, this element is the number of message rows after which the adapter issues a commit command to the external data source and a GD commit to the stream to which the adapter is attached. 
If the adapter is running without GD mode, BatchSize is the number of message rows after which the adapter issues a commit command to the external data source. The default value is 1. 
Increasing this value improves performance at the expense of latency. It also increases memory consumption in the smart data streaming server because the uncommitted rows need to be preserved for redelivery in case of failure. 
If the smart data streaming Subscriber module EnableGDMode element is set to true, set either BatchSize or the BatchPeriod property to greater than 0. If neither property is set to greater than 0, a warning is sent and BatchSize is set to 1. |
| BatchPeriod | Type: integer  
(Advanced) If the adapter is running in GD mode, this element is the number of seconds after which the adapter issues a commit command to the external data source and a GD commit to the stream to which the adapter is attached. 
If the adapter is running without GD mode, BatchPeriod is the number of seconds after which the adapter issues a commit command to the external data source. The default value is 0. 
Increasing this value improves performance at the expense of latency. It also increases memory consumption in the smart data streaming server because the uncommitted rows need to be preserved for redelivery in case of failure. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileSizeLimit</td>
<td>Type: integer</td>
</tr>
<tr>
<td></td>
<td>(Optional) The maximum size, in bytes, of the output file. If this property is set, the adapter starts writing a new file every time the size of the current output file becomes greater than this value. The files are named <code>&lt;file-name&gt;.001</code>, <code>&lt;file-name&gt;.002</code>, and so on, where <code>&lt;file-name&gt;</code> is the value of the <code>File</code> element.</td>
</tr>
<tr>
<td></td>
<td>If you do not specify a value or you specify a value of 0 for this property, the output file does not have a size limit.</td>
</tr>
<tr>
<td></td>
<td>No default value.</td>
</tr>
<tr>
<td>TimeBasedRotate</td>
<td>Type: boolean</td>
</tr>
<tr>
<td></td>
<td>(Optional) Indicates whether to rotate files at predefined intervals.</td>
</tr>
<tr>
<td></td>
<td>The default value is false.</td>
</tr>
<tr>
<td>TimeBasedRotateInterv</td>
<td>Type: interval</td>
</tr>
<tr>
<td></td>
<td>(Optional) The amount of time, in seconds, to wait between file rotations.</td>
</tr>
<tr>
<td></td>
<td>The default value is 24 hours.</td>
</tr>
<tr>
<td>TimeBasedRotateStartAt</td>
<td>Type: time</td>
</tr>
<tr>
<td></td>
<td>(Optional) The time of the first file rotation. The supported formats are:</td>
</tr>
<tr>
<td></td>
<td>● <code>yyyy-MM-DD HH:mm:ss.SS z</code></td>
</tr>
<tr>
<td></td>
<td>● <code>yyyy-MM-DD HH:mm:ss</code></td>
</tr>
<tr>
<td></td>
<td>● <code>yyyy-MM-DD HH:mm:ss</code></td>
</tr>
<tr>
<td></td>
<td>● <code>yyyy-MM-DD HH:mm z</code></td>
</tr>
<tr>
<td></td>
<td>● <code>yyyy-MM-DD HH:mm</code></td>
</tr>
<tr>
<td></td>
<td>The default value is 0:00 UTC.</td>
</tr>
<tr>
<td></td>
<td>If you do not specify a time zone, the adapter defaults to UTC.</td>
</tr>
<tr>
<td>TimestampInFilename</td>
<td>Type: boolean</td>
</tr>
<tr>
<td></td>
<td>(Optional) Indicates whether to append the system time, in UTC, to the output file name when a new output file is created.</td>
</tr>
<tr>
<td></td>
<td>The default value is false.</td>
</tr>
<tr>
<td>XML Element</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| **TimestampInFilename**           | **Type:** string  
(Optional) The timestamp format that gets appended to the output file name. Valid formats include any formats accepted by the Java SimpleDateFormat class. For example, `yyyy-MM-dd_HH-mm-ss`.  
On Windows, the following symbols are not permitted in the file name: `/:*'?"<>`.  
The default timestamp format is `yyyyMMdd_HHmmss`.                                                                                                                                                                                                                                                                                                                                                     |
| **HDFSReplaceDataNodeOnFailureEnable** | **Type:** boolean  
(Optional; applicable only for use with HDFS folders) Enable this property to provide an alternate value for the `dfs.client.block.write.replace-datanode-on-failure.enable` property. Use only with Hadoop 2.2.0 or higher. For additional information on configuring this property, see the Hadoop documentation.  
By default, this property is commented out and the adapter inherits the default value for the `dfs.client.block.write.replace-datanode-on-failure.enable` property on the Hadoop client side.                                                                                                                                                                                                 |
| **HDFSReplaceDataNodeOnFailurePolicy** | **Type:** string  
(Optional; applicable only for use with HDFS folders) Enable this property to provide an alternate value for the `dfs.client.block.write.replace-datanode-on-failure.policy` property. Use only with Hadoop 2.2.0 or higher. For additional information on configuring this property, see the Hadoop documentation.  
By default, this property is commented out and the adapter inherits the default value for the `dfs.client.block.write.replace-datanode-on-failure.policy` property on the Hadoop client side.                                                                                                                                                      |
| **HDFSReplication**               | **Type:** int  
(Optional; applicable only for use with HDFS folders) Enable this property to provide an alternate value for `dfs.replication` property. Use only with Hadoop 2.2.0 or higher.  
If you are using a Hadoop system with less than three nodes, set this property accordingly. For additional information on configuring this property, see the Hadoop documentation.  
By default, this property is commented out and the adapter inherits the default value for the `dfs.replication` property on the Hadoop client side.                                                                                                                                                                                                 |

---

**SAP HANA Smart Data Streaming: Building Custom Adapters**

**Smart Data Streaming Adapter Toolkit**
### 2.4.1.3 FTP Input Transporter Module Properties

The FTP Input transporter reads binary data from files on an FTP server, wraps it up with ByteBuffer, and sends it to the next module that is configured in the adapter configuration file. Set values for this transporter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Type: string&lt;br&gt;(Required) The server name or IP address of the FTP server to which you are connecting.</td>
</tr>
<tr>
<td>Port</td>
<td>Type: integer&lt;br&gt;(Required) The port address for the FTP server to which you are connecting. The default value is 21.</td>
</tr>
<tr>
<td>LoginType</td>
<td>Type: enum&lt;br&gt;(Required) The login type for the FTP server: normal or anonymous.</td>
</tr>
<tr>
<td>User</td>
<td>Type: string&lt;br&gt;(Required if LoginType is set to normal) The login account for the FTP server.</td>
</tr>
<tr>
<td>Password</td>
<td>Type: string&lt;br&gt;(Required if LoginType is set to normal) The login password for the FTP server.</td>
</tr>
<tr>
<td>FtpFilePath</td>
<td>Type: string&lt;br&gt;(Required) The absolute path to the data files in the FTP server.</td>
</tr>
<tr>
<td>FtpFileName</td>
<td>Type: string&lt;br&gt;(Required) The data files in the FTP server.</td>
</tr>
<tr>
<td>MaxBlockSize</td>
<td>Type: int&lt;br&gt;(Required) The maximum size of any data blocks transferred from the FTP server. The default value is 2048.</td>
</tr>
<tr>
<td>TransferMode</td>
<td>Type: string&lt;br&gt;(Required) The transfer mode for the FTP connection: active or passive. The default value is active.</td>
</tr>
<tr>
<td>RetryPeriod</td>
<td>Type: second&lt;br&gt;(Required) The period of time, in seconds, during which to try and reconnect to the FTP server if you disconnect unexpectedly. The default value is 30.</td>
</tr>
</tbody>
</table>
### 2.4.1.4 FTP Output Transporter Module Properties

The FTP Output transporter obtains data from the previous module configured in the adapter configuration file, and saves it to files on the FTP server. Set values for this transporter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Type: string</td>
</tr>
<tr>
<td></td>
<td>(Required) The server name or IP address of the FTP server to which you are connecting.</td>
</tr>
<tr>
<td>Port</td>
<td>Type: integer</td>
</tr>
<tr>
<td></td>
<td>(Required) The port address for the FTP server to which you are connecting. The default value is 21.</td>
</tr>
<tr>
<td>LoginType</td>
<td>Type: enum</td>
</tr>
<tr>
<td></td>
<td>(Required) The login type for the FTP server. There are two valid types: normal and anonymous.</td>
</tr>
<tr>
<td>User</td>
<td>Type: string</td>
</tr>
<tr>
<td></td>
<td>(Required if LoginType is set to normal) The login account for the FTP server.</td>
</tr>
<tr>
<td>Password</td>
<td>Type: string</td>
</tr>
<tr>
<td></td>
<td>(Required if LoginType is set to normal) The login password for the FTP server.</td>
</tr>
<tr>
<td>FtpFilePath</td>
<td>Type: string</td>
</tr>
<tr>
<td></td>
<td>(Required) The absolute path to the data files in the FTP server.</td>
</tr>
<tr>
<td>FtpFileName</td>
<td>Type: string</td>
</tr>
<tr>
<td></td>
<td>(Required) The data files in the FTP server.</td>
</tr>
<tr>
<td>MaxBlockSize</td>
<td>Type: int</td>
</tr>
<tr>
<td></td>
<td>(Required) The maximum data block size to transfer to the FTP server. The default value is 2048.</td>
</tr>
</tbody>
</table>
XML Element | Description
--- | ---
Overwrite | Type: boolean  
(Required) If set to true, the transporter overwrites the file on the FTP server, if it exists. If this element is set to false, the transporter appends the output to the end of the existing file.  
The default value is false.
TransferMode | Type: string  
(Required) The transfer mode for the FTP connection. There are two valid values: active or passive. The default value is active.
RetryPeriod | Type: second  
(Required) The period of time, in seconds, during which to try and reconnect to the FTP server if you disconnect unexpectedly. The default value is 30.
RetryNumber | Type: integer  
(Required) The number of times to attempt to reconnect to the FTP server if you disconnect unexpectedly. The default value is 0. A value of -1 will make the module retry forever, with no stops.

### 2.4.1.5 HTTP Output Transporter Module Properties

The HTTP Output transporter obtains stream data from smart data streaming and sends it to an HTTP server. Set values for this transporter in the adapter configuration file.

**Note**

If the network is configured to use proxy servers, set the `no_proxy` environment variable on all publisher and subscriber machines to ensure successful publishing and subscribing of data.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| ProxyHost | Type: string  
(Advanced) The proxy server hostname. No default value. |
| ProxyPort | Type: integer  
(Advanced) The proxy server port. No default value. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RetryNumber</td>
<td>Type: integer                                                                                                                                             (Optional) The number of times to try reconnecting to the HTTP server if the adapter disconnects unexpectedly. The default value is 0. A value of -1 will make the module retry forever, with no stops.</td>
</tr>
<tr>
<td>contentType</td>
<td>Type: string                                                                                                                                             (Optional) The content type of the HTTP request. Valid values include: text/plain, text/xml, text/html, application/atom+xml, application/json, application/xml. The default value is text/plain.</td>
</tr>
<tr>
<td>threadNumber</td>
<td>Type: integer                                                                                                                                             (Optional) The number of threads to try to connect to the HTTP server. If the EspSubscriber does not enable guaranteed delivery (GD) mode, the default value is the CPU core number of the machine running the adapter. If the EspSubscriber enables GD mode, the default value is 1 and cannot be changed. If the value is less than or equal to 0, the adapter cannot start.</td>
</tr>
<tr>
<td>BodyCharset</td>
<td>Property ID: bodyCharset                                                                                                                                Type: string                                                                                                                                             (Optional) The name of a supported character set. The default value is UTF-8.</td>
</tr>
<tr>
<td>UrlColumn</td>
<td>Type: integer                                                                                                                                             (Dependent required) The smart data streaming stream column index, which the adapter is attached to, containing the URL to use for sending HTTP requests. This property is required if RequestUrl is empty.</td>
</tr>
<tr>
<td>UsernameColumn</td>
<td>Type: integer                                                                                                                                             (Optional) The smart data streaming stream column index, which the adapter is attached to, containing the username to use if the HTTP server requires authentication.</td>
</tr>
<tr>
<td>XML Element</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| PasswordColumn      | Type: integer  
(Optional) The smart data streaming stream column index, which the adapter is attached to, containing the password to use if the HTTP server requires authentication.                                      |
| BodyColumn          | Type: integer  
(Required) The smart data streaming stream column index, which the adapter is attached to, containing the message body content for the HTTP request.                                                          |
| RequestUrl          | Type: string  
(Optional) The URL to use for sending HTTP requests. This URL is used if the `UrlColumn` element is not specified, or if the smart data streaming stream does not have a URL column.                      |
| RequestUsername     | Type: string  
(Optional) The username to use if the HTTP server requires authentication. This username is used if the `UsernameColumn` element is not specified, or if the smart data streaming stream does not have a username column.         |
| RequestPassword     | Type: string  
(Optional) The password to use if the HTTP server requires authentication. This password is used if the `PasswordColumn` element is not specified, or if smart data streaming stream does not have a password column.  
This property includes an encrypted attribute, which indicates whether the password value is encrypted. |
| RequestBody         | Type: string  
(Optional) The message body content for the HTTP request. This value is used if the `BodyColumn` element is not specified, or if the smart data streaming stream does not have a body column.                |
| RSAKeyStore         | Type: string  
(Dependent required) The location of an RSA keystore file that contains the key used to encrypt and decrypt the password set in the RequestPassword element. This element is required if the password value set in RequestPassword is encrypted. |
### 2.4.1.6 JDBC Input Transporter Module Properties

The JDBC Input transporter reads database records using JDBC and sends them to the next module specified in the adapter configuration file. Set values for this transporter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| **RSAPassword**   | **Type:** string  
                    (Dependent required) Stores the password to the RSA key-store file specified in the RSAKeyStore element. This element is required if the password value set in RequestPassword is encrypted. |
| **RSAKeyStoreAlias** | **Type:** string  
                    (Dependent required) The location of the keystore alias. This element is required if the password value set in RequestPassword is encrypted. |

### XML Element | Description
--- | ---
**Host** | **Type:** string  
(Required) The server name of the database to which you are connecting the adapter.

**Port** | **Type:** integer  
(Required) The port number for connecting to the database server.

**Username** | **Type:** string  
(Required) The user name you are using to connect to the database server.

**Password** | **Type:** string  
(Required) The password for connecting to the database server. Includes an "encrypted" attribute that indicates whether the password value is encrypted.

**DbName** | **Type:** string  
(Required) The database to which you want to connect.

**DBType** | **Type:** string  
(Required) The database type of the database to which you want to connect.
### 2.4.1.7 JDBC Output Transporter Module Properties

The JDBC Output transporter obtains data from the previous module specified in the adapter configuration file and writes it into a database table using JDBC. Set values for this transporter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Type: string (Required) The server name of the database to which you are connecting the adapter.</td>
</tr>
<tr>
<td>XML Element</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| **Port**    | Type: integer  
(Required) The port number for connecting to the database server. |
| **Username**| Type: string  
(Required) The username you are using to connect to the database server. |
| **Password**| Type: string  
(Required) The password for connecting to the database server. Includes an "encrypted" attribute that indicates whether the password value is encrypted. |
| **DbName**  | Type: string  
(Required) The database name of the database to which you want to connect. |
| **DBType**  | Type: string  
(Required) The database type of the database to which you want to connect. |
| **DbDriver**| Type: string  
(Required) The JDBC driver class for your JDBC driver. |
| **Table**   | Type: string  
(Optional) The name of the table in the target database to which you want the adapter to write. |
| **SqlInsert**| Type: string  
(Optional) The SQL clause you want the adapter to execute. No default value.  
Set either the Table or SqlInsert element. If you define both elements, the adapter uses only the SqlInsert element. |
| **RSAKeyStore**| Type: string  
(Dependent required) The location of an RSA keystore file that contains the key used to encrypt or decrypt the password set in the Password element. This element is required if the password value is encrypted. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| RSAKeyStorePassword    | Type: string  
(Dependent required) Stores the password to the RSA key-store file specified in the RSAKeyStore element. This element is required if the password value is encrypted. |
| RSAKeyStoreAlias       | Type: string  
(Dependent required) The keystore alias. This element is required if the password value is encrypted. |
| BatchSize              | Type: integer  
(Optional) If the adapter is running in GD (guaranteed delivery) mode, this element is the number of message rows after which the adapter issues a commit command to the external data source and a GD commit to the stream to which the adapter is attached.  
If the adapter is running without GD, this element is the number of message rows after which the adapter issues a commit command to the external data source. The default value is 1.  
Increasing this value improves performance at the expense of latency. It also increases memory consumption in the smart data streaming server because the uncommitted rows need to be preserved for redelivery in case of failure.  
If running this adapter in GD mode, set either this element, or the BatchPeriod element to greater than 0. If neither is set to greater than 0, a warning is sent and this property is set to 1. |
2.4.1.8 JMS Input Transporter Module Properties

The JMS Input transporter receives JMS messages from a JMS server, and sends it to the next module that is configured in the adapter configuration file. Set values for this transporter in the adapter configuration file.

The transporter sends different Java objects to the next module depending on the message type it receives:

<table>
<thead>
<tr>
<th>Message Type Received</th>
<th>Java Object to Send</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT (TextMessage)</td>
<td>String</td>
</tr>
<tr>
<td>OBJARRAY (ObjectMessage)</td>
<td>Object</td>
</tr>
<tr>
<td>MAP (MapMessage)</td>
<td>String</td>
</tr>
</tbody>
</table>

**BatchPeriod**

Type: integer

(Optional) If the adapter is running in GD mode, this element is the number of seconds after which the adapter issues a commit command to the external data source and a GD commit to the stream to which the adapter is attached.

If the adapter is running without GD mode, this element is the number of seconds after which the adapter issues a commit command to the external data source. The default value is 0.

Increasing this value improves performance at the expense of latency. It also increases memory consumption in the smart data streaming server because the uncommitted rows need to be preserved for redelivery in case of failure.

If running the adapter in GD mode, set either this or the BatchSize property to greater than 0. If not, a warning is sent and this property is set to 1.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionFactory</td>
<td>Type: string</td>
</tr>
<tr>
<td></td>
<td>(Required) The JNDI name for the JMS server connection factory. Consult your third-party vendor documentation for specific formats. Here are some examples:</td>
</tr>
<tr>
<td>ActiveMQ</td>
<td>ConnectionFactory</td>
</tr>
<tr>
<td>TIBCO</td>
<td>QueueConnectionFactory</td>
</tr>
<tr>
<td>WebSphere MQ</td>
<td>MyMQConnFactory</td>
</tr>
<tr>
<td>No default value.</td>
<td></td>
</tr>
<tr>
<td>XML Element</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| JndiContextFactory  | Type: string  
(Required) Context factory for JNDI context initialization. Consult your third-party vendor documentation for specific formats. Here are some examples:  
**ActiveMQ**  
org.apache.activemq.jndi.ActiveMQInitialContextFactory  
**TIBCO**  
com.tibco.tibjms.naming.TibjmsInitialContextFactory  
**WebSphere MQ**  
com.sun.jndi.fscontext.RefFSContextFactory  
No default value. |
| JndiUrl             | Type: string  
(Required) The JNDI URL. Consult your third-party vendor documentation for specific formats. Here are some examples:  
**ActiveMQ**  
tcp://server:61616  
**TIBCO**  
tibjmsnaming://server:7222  
**WebSphere MQ**  
file:/var/mqm/jndi/  
WebSphere MQ requires a separate naming server to be configured with it. By default, WebSphere MQ only provides a file-based naming server.  
You can also enable automatic reconnection here. Consult your third-party vendor documentation for specific formats. Here is an example for ActiveMQ, failover:tcp://<hostname>:<port-number>.  
No default value. |
| Destination Type    | Type: string  
(Required) The destination type. Valid values are QUEUE and TOPIC. The default value is QUEUE. |
| DestinationName     | Type: string  
(Required) The destination name. No default value. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| MessageType       | Type: string
(Required) The message type for the JMS transporter to process:
- TEXT – to receive and send messages in text string.
- OBJARRAY – to receive and send messages in custom format.
- MAP – to receive and send messages in sets of name-value pairs.
No default value. |
| SubscriptionMode | Type: string
(Optional) The subscription mode for TOPIC (see the Destination Type element). Valid values are DURABLE and NONDURABLE.
The default value is NONDURABLE. |
| ScanDepth        | Type: integer
(Optional) The depth of the schema discovery. The adapter reads the number of rows specified by this element value when discovering the input data schema.
The default value is 3. |
| ClientID         | Type: string
(Required for DURABLE subscription mode) The client identifier for a JMS client. Can be any string, but must be unique for each topic. No default value.
Example: client1 |
| SubscriptionName | Type: string
(Required for DURABLE subscription mode) A unique name identifying a durable subscription. Can be any string, but must be unique within a given client ID. No default value.
Example: subscription1 |
| Username         | Type: string
(Advanced) A valid username for connecting to a specific JMS server. No default value. |
### Password

**Type:** `string`

(Advanced) A valid password for connecting to a specific JMS server. Contains an encrypted attribute that determines whether the `Password` value is encrypted. No default value.

### 2.4.1.9 JMS Output Transporter Module Properties

The JMS Output transport obtains data from the previous module that is configured in the adapter configuration file, wraps it up, and sends it to a JMS server. Set values for this transporter in the adapter configuration file.

The transporter sends different JMS messages depending on the datatype it receives from the previous module:

<table>
<thead>
<tr>
<th>Datatype Received</th>
<th>JMS Message Type to Send</th>
</tr>
</thead>
<tbody>
<tr>
<td>List&lt;String&gt;</td>
<td>TextMessage</td>
</tr>
<tr>
<td>AepRecord</td>
<td>ObjectMessage</td>
</tr>
<tr>
<td>Object (others)</td>
<td>ObjectMessage</td>
</tr>
<tr>
<td>List&lt;String&gt;</td>
<td>MapMessage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionFactory</td>
<td>Type: <code>string</code></td>
</tr>
</tbody>
</table>

(Required) The JNDI name for the JMS server connection factory. Consult your third-party vendor documentation for specific formats. Here are some examples:

- **ActiveMQ**: ConnectionFactory
- **TIBCO**: QueueConnectionFactory
- **WebSphere MQ**: MyMQConnFactory

No default value.
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| JndiContextFactory     | Type: string 
(Required) Context factory for JNDI context initialization. Consult your third-party vendor documentation for specific formats. Here are some examples:  
**ActiveMQ**  org.apache.activemq.jndi.ActiveMQInitialContextFactory  
**TIBCO**  com.tibco.tibjms.naming.TibjmsInitialContextFactory  
**WebSphere MQ**  com.sun.jndi.fscontext.RefFSContextFactory  
No default value. |
| JndiUrl                 | Type: string 
(Required) The JNDI URL. Consult your third-party vendor documentation for specific formats. Here are some examples:  
**ActiveMQ**  tcp://server:61616  
**TIBCO**  tibjmsnaming://server:7222  
**WebSphere MQ**  file:/var/mqm/jndi/  
WebSphere MQ requires a separate naming server to be configured with it. By default, WebSphere MQ only provides a file-based naming server.  
You can also enable automatic reconnection here. Consult your third-party vendor documentation for specific formats. Here is an example for ActiveMQ, failover:  
tcp://<hostname><port-number>.  
No default value. |
| Destination Type       | Type: string 
(Required) The destination type. Valid values are QUEUE and TOPIC. The default value is QUEUE. |
| DestinationName        | Type: string 
(Required) The destination name. No default value. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| MessageType     | **Type:** string  
    *(Required)* The message type for the JMS transporter to process:  
    - TEXT – to receive and send messages in text string.  
    - OBJARRAY – to receive and send messages in custom format.  
    - MAP – to receive and send messages in sets of name-value pairs.  
    No default value. |
| DeliveryMode    | **Type:** string  
    *(Optional)* The delivery mode type. Valid values are PERSISTENT and NON_PERSISTENT. Default value is PERSISTENT. |
| BatchSize       | **Type:** integer  
    *(Advanced)* If the adapter is running in GD (guaranteed delivery) mode, this element is the number of message rows after which the adapter issues a *commit* command to the external data source and a GD commit to the stream to which the adapter is attached.  
    If the adapter is running without GD mode, *BatchSize* is the number of message rows after which the adapter issues a *commit* command to the external data source. The default value is 1.  
    Increasing this value improves performance at the expense of latency. It also increases memory consumption in the smart data streaming server because the uncommitted rows need to be preserved for redelivery in case of failure.  
    If the smart data streaming Subscriber module *EnableGDMode* element is set to true, set either *BatchSize* or the *BatchPeriod* property to greater than 0. If neither property is set to greater than 0, a warning is sent and *BatchSize* is set to 1. |
### XML Element

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| BatchPeriod   | Type: integer  
                (Advanced) If the adapter is running in GD mode, this element is the number of seconds after which the adapter issues a commit command to the external data source and a GD commit to the stream to which the adapter is attached. If the adapter is running without GD mode, BatchPeriod is the number of seconds after which the adapter issues a commit command to the external data source. The default value is 0. Increasing this value improves performance at the expense of latency. It also increases memory consumption in the smart data streaming server because the uncommitted rows need to be preserved for redelivery in case of failure. |
| Username      | Type: string  
                (Advanced) A valid username for connecting to a specific JMS server. No default value. |
| Password      | Type: string  
                (Advanced) A valid password for connecting to a specific JMS server. Contains an encrypted attribute that determines whether the Password value is encrypted. No default value. |

### 2.4.1.10 Kafka Input Transporter Module Properties

The Kafka Input transporter acts as a Kafka consumer. It reads data from a Kafka server and sends string data to smart data streaming.

This transporter supports schema discovery.
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| TopicPartition    | Type: string  
(Optional) The topics or topic partitions from which the adapter inputs data. You can use more than one topic or topic partition in this parameter.  
For example:  
1. For topic0 and topic1, with partitions 0 and 1, you can configure the adapter to subscribe to a topic’s partition using the following syntax: `topic0:0,topic1:1`  
2. To configure the adapter to subscribe to topic0 and topic1: `topic0,topic1`  
Don’t set `TopicPartition` and `TopicPattern` simultaneously because they conflict. `TopicPartition` has priority over `TopicPattern`.  
Also, don’t set two topics to the same partition. For example, this configuration is invalid: `topic:0,topic1:0` |
| TopicPattern      | Type: string  
(Optional) A regular expression pattern. The adapter inputs data from topics that match this pattern.  
Don’t set `TopicPattern` and `TopicPartition` simultaneously because they conflict. `TopicPartition` has priority over `TopicPattern`. |
| Offset            | Type: integer  
(Optional) The offset for the first partition from which the adapter receives input.  
This element is skipped when you are inputting data from a topic instead of a partition. |
| BootstrapServers  | Type: string  
(Optional) The names of the bootstrap servers. You can also set this in the consumer property file using `bootstrap.servers`.  
If you set this element in both files, `BootstrapServers` (in the .XML file) overwrites `bootstrap.servers`. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| FromBeginning     | Type: boolean  
(Optional) Indicates whether the adapter reads all the data or only newly generated data in a topic. If set to true, all of the data in the topic is read. If set to false, only newly generated data in the topic is read.  
This property is skipped when you are inputting data from a topic instead of a partition.  
The default value is false. However, if you are performing schema discovery, the default value is true. |
| GroupId           | Type: string  
(Optional) The GroupId name when the adapter connects to a server.  
You can also set this in the consumer property file as group.id. If you set both, GroupId (in the .XML file) overwrites group.id.  
This property is skipped in discovery mode because the adapter uses its own group ID to perform schema discovery. |
| ConsumerProperties| Type: string  
(Optional) The filepath to the Kafka consumer property file.  
You can set additional properties in this file. See the Kafka documentation for more details. |
| PollInterval      | Type: integer  
(Optional) The time period, in milliseconds, to poll the server. The default value is 1000. |
| ScanDepth         | Type: integer  
(Optional) The depth of the schema discovery. The adapter reads the number of rows specified by this parameter value when discovering the input data schema.  
The default value is 3. |
| GDMaxRetryTimes   | Type: integer  
(Optional) The number of times the adapter tries resending data in case of failure when guaranteed delivery (GD) is enabled.  
The default value is 30. |
2.4.1.11 Kafka Output Transporter Module Properties

The Kafka Output transporter acts as a Kafka producer. It reads data from smart data streaming and writes string data asynchronously to a Kafka server.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| BootstrapServers  | Type: string  
(Optional) The names of the bootstrap servers. You can also use the bootstrap.servers property in the producer property file to set up a broker server. See the Kafka documentation for additional details.  
If both BootstrapServers and bootstrap.servers properties are set, BootstrapServers overwrites bootstrap.servers. |
| TopicPartition    | Type: string  
(Required) The Kafka topic or topic partition from which the adapter publishes data.  
For example:  
1. To configure the adapter to publish to partition 0 of topic1: `topic1:0`  
2. To configure the adapter to publish to topic0: `topic0` |
| ProducerProperties| Type: string  
(Optional) The filepath to the Kafka producer property file. You can set additional properties in this file. See the Kafka documentation for more details. |
| BatchSize         | Type: integer  
(Optional) After how many rows the adapter performs a flush in guaranteed delivery (GD) mode. The default value is 1. |
| BatchPeriod       | Type: integer  
(Optional) After how many seconds the adapter performs a flush in guaranteed delivery (GD) mode. The default value is 0. |
# 2.4.1.12 Socket Input Transporter Module Properties

The Socket Input transporter reads binary data from the socket interface, wraps it with ByteBuffer, and sends it to the next module that is configured in the adapter configuration file. Set values for this transporter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Host</strong></td>
<td>Type: string&lt;br&gt;(Required if EpFile is set to null) The socket server name. No default value.</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>Type: integer&lt;br&gt;(Required if EpFile is set to null) The socket server port. If you set this to -1, the adapter reads from the ephemeral port file, which is specified in the EpFile element. The default value is 12345.</td>
</tr>
<tr>
<td><strong>EpFile</strong></td>
<td>Type: string&lt;br&gt;(Required if Host and Port are set to null) The file that contains the socket server name/IP and port number. This file resides in the <code>&lt;sandbox-base-directory&gt;/adapters/&lt;workspace-name&gt;</code> or <code>$STREAMING_HOME/adapters/framework</code> directory or one of their subdirectories. If the file is in a different location, the adapter will not be able to successfully connect to smart data streaming. Use only host and port information, on separate lines, in this file. For example, the syntax is:</td>
</tr>
<tr>
<td></td>
<td>host=hostName&lt;br&gt;port=portNumber&lt;br&gt;No default value.</td>
</tr>
<tr>
<td><strong>Retryperiod</strong></td>
<td>Type: integer&lt;br&gt;(Advanced) The time period for attempting to reestablish an outgoing connection, in milliseconds. The default value is 0.</td>
</tr>
<tr>
<td><strong>RetryNumber</strong></td>
<td>Type: integer&lt;br&gt;(Required) The number of times to attempt to reestablish the outgoing connection if you disconnect unexpectedly. The default value is 30. A value of -1 will make the module retry forever, with no stops.</td>
</tr>
</tbody>
</table>
### BlockSize

- **Type:** integer
- (Advanced) The size of the data block when transporting data from the socket server to the transporter. The default value is 1024.

### KeepAlive

- **Type:** boolean
- (Advanced) If set to true, the adapter disconnects from the socket server if there are no data transports for the duration of time specified in your router configuration. For example, if you set your router configuration to two hours and there are no messages during that time, the adapter disconnects from the socket server. The default value is false.

## 2.4.1.13 Socket Output Transporter Module Properties

The Socket Output transporter obtains data from the previous module configured in the adapter configuration file, and outputs it using the socket interface. Set values for this transporter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Host**    | Type: string
- (Required if EpFile is set to null) The socket server name. No default value. |
| **Port**    | Type: integer
- (Required if EpFile is set to null) The socket server port. If you set this to -1, the adapter reads from the ephemeral port file, which is specified in the EpFile element. The default value is 12345. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| EpFile           | Type: string                                                                                                                                  
|                  | *(Required if Host and Port are set to null)* The file that contains the socket server name/IP and port number. This file resides in the `<sandbox-base-directory>/adapters/<workspace-name>` or $STREAMING_HOME/adapters/framework directory or one of its subdirectories. If the file is in a different location, the adapter cannot connect to smart data streaming. Use only host and port information, on separate lines, in this file. For example, the syntax is:  
|                  | host=hostName  
|                  | port=portNumber  
|                  | No default value.                                                                                                                            |
| Retryperiod      | Type: integer                                                                                                                                  
|                  | *(Advanced)* The time period for attempting to reestablish an outgoing connection, in milliseconds. The default value is 0.                                                                                |
| RetryNumber      | Type: integer                                                                                                                                  
|                  | *(Required)* The number of times to attempt to reestablish the outgoing connection if you disconnect unexpectedly. The default value is 30. A value of -1 will make the module retry forever, with no stops. |
| KeepAlive        | Type: boolean                                                                                                                                  
|                  | *(Advanced)* If set to true, the adapter disconnects from the socket server if there are no data transports for the duration of time specified in your router configuration. For example, if you set your router configuration to two hours and there are no messages during that time, the adapter disconnects from the socket server. The default value is false. |
2.4.2 Building a Custom Transporter Module

Use the smart data streaming adapter toolkit to build a custom transporter module to use within the adapter instance of your choice.

Prerequisites

(Optional) See the $STREAMING_HOME/adapters/framework/examples/src directory for source code for sample transporters.

Procedure

1. Create a class that extends the com.sybase.esp.adapter.framework.module.Transporter Java class that is included with the adapter toolkit.
2. Implement the init() function.
   Prepare your input or output transporter module for the actions it is responsible for performing. For example, create a database connection or obtain properties from the adapter configuration file.
3. Implement the start() function.
   Perform any necessary tasks when the adapter is started.
4. Implement the execute() function.
   When the adapter framework calls this method, it is expected to run continuously until the adapter is requested to stop or until the adapter completes its work. Therefore, the code excerpt below might be found within a loop, or inside a callback method invoked by the transport when an event occurs, or inside a listener monitoring transport events.

AepRecord is a single record or row in smart data streaming format and has an operation code that can be set. Its default operation code is INSERT. Change this code based on your needs. AdapterRow represents records as they flow from one module to the next. You can add multiple records as objects within a List of a single AdapterRow object. The AdapterRow has a timestamp, and block flags that control how its records are communicated to and from smart data streaming. See Event Blocks in the SAP HANA Smart Data Streaming: CCL Reference for additional details.

The actions performed by this function depend on whether the transporter is input (data source) or output (data sink). For example, for an input transporter that gets data from "myDataSource", the execute() function might look like this:

```java
public void execute() throws Exception {
    String value = myDataSource.getNextValue();
    AepRecord record = new AepRecord();
    record.getValues().add(value);
    AdapterRow row = utility.createRow(record);
    utility.sendRow(row);
}
```
For an output transporter that sends data to "myDataSink", the execute() function might look like this:

```java
public void execute() throws Exception {
    AdapterRow row = utility.getRow();
    if(row != null) {
        AepRecord record = (AepRecord)row.getData(0);
        if(record != null) {
            String value = record.getValues().toString();
            myDataSink.send(value);
        }
    }
}
```

The difference between input and output transporters is that input transporters call utility.sendRow() to send data to a formatter or smart data streaming publisher, while output transporters call utility.getRow() to obtain data from a formatter or smart data streaming subscriber.

For transporters that operate in streaming mode, call utility.sendRowsBuffer() (input) and utility.getRowsBuffer() (output).

See the $STREAMING_HOME/adapters/framework/examples/src directory for source code for sample transporters.

5. Implement the stop() function.
   Perform any necessary tasks when the adapter is stopped.

6. Implement the destroy() function.
   Perform any cleanup tasks for your input or output transporter.

7. (Optional) Call one of the following functions within the functions listed in the steps above:
   - utility.getParameters() – to get parameters that are defined in the adapter configuration file.
   - utility.sendRow() – to send data to the next module that is defined in the adapter configuration file.
   - utility.getRow() – to obtain data from the previous module that is defined in the adapter configuration file.
   - utility.isStopRequested() – to determine whether a stop command has been issued.

8. Register the implemented Java class to $STREAMING_CUSTOM_ADAPTERS_HOME/config/custommodulesdefine.xml. For example:

```xml
<TransporterDefn>
   <Name>MyOutputTransporter</Name>
   <Class>com.my.MyOutputTransporter</Class>
   <InputData>String</InputData>
</TransporterDefn>
```

9. Add the schema definitions for any unique parameters of the newly created module to the $STREAMING_CUSTOM_ADAPTERS_HOME/config/parametersdefine.xsd file.

   If any of the parameters for the newly created module are the same as parameters for the standard transporter modules, you don’t need to add schema definitions for these parameters.

10. Copy the .jar file containing the Java class you previously implemented and any other .jar files used by the custom adapter to $STREAMING_CUSTOM_ADAPTERS_HOME/libj.

11. (Optional) Start the adapter instance by issuing the following, where <config-file> is the adapter configuration file in which you specified the adapter instance using the newly created transporter module:

   - For Windows, $STREAMING_HOME/adapters/framework/bin/start.bat <config-file>
For Linux, `$STREAMING_HOME/adapters/framework/bin/start.sh <config-file>`

where `<config-file>` is the adapter configuration file in which you specified the adapter instance using the newly created transporter module.

12. (Optional) Stop the adapter instance by issuing, where `<config-file>` is the adapter configuration file in which you specified the adapter instance using the newly created transporter module:

- For Windows, `$STREAMING_HOME/adapters/framework/bin/stop.bat <config-file>`
- For Linux, `$STREAMING_HOME/adapters/framework/bin/stop.sh <config-file>`

**Example**

See `$STREAMING_HOME/adapters/framework/examples` for additional details and transporter examples, as well as `$STREAMING_HOME/adapters/framework/examples/src` for the source code for these examples.

**Next Steps**

Create an adapter configuration (.xml) file to define which adapter instance uses this newly created transporter module.

**Task overview:** Create a Custom Adapter [page 18]

**Next task:** Building a Custom Formatter Module [page 98]

**Related Information**

- Accessing Adapter Toolkit API Reference Information [page 19]
- Transporters Currently Available from SAP [page 21]

### 2.5 Formatter Modules

A formatter module converts between the data format of the transporter module and smart data streaming.

SAP HANA smart data streaming supports two types of formatters: row-based and stream-based formatters.

Row-based formatters obtain and output data in row format. They work with AdapterRow instances, which are containers for one or more records or rows as they flow from one module (transporter, formatter, or smart data streaming connector) to the next. You can add multiple records as objects within a List of a single AdapterRow object. The AdapterRow has a timestamp and block flags that control how its records are communicated to and from smart data streaming. See *Event Blocks* in the *SAP HANA Smart Data Streaming: CCL Reference* for additional details.
Stream-based formatters deal with streaming data. These formatters work with ByteStream instances, which represent a continuous stream of data.

In this section:

- **Formatters Currently Available from SAP [page 60]**
  The adapter framework provides numerous formatter modules that come standard with smart data streaming. You can reuse these modules to create a custom adapter instance.

- **Datatype Mapping for Formatters [page 97]**
  Mapping information for smart data streaming to Java datatypes and Java to smart data streaming datatypes.

- **Building a Custom Formatter Module [page 98]**
  Use the smart data streaming adapter toolkit to build a custom formatter module to use within the adapter instance of your choice.

- **Datatype Mapping for JMS Map Formatter [page 101]**
  Mapping information for smart data streaming to JMS MapMessage datatypes and JMS MapMessage to smart data streaming datatypes.

Related Information

- Transporter Modules [page 20]
- EspConnector Modules [page 108]
- Accessing Adapter Toolkit API Reference Information [page 19]
- Create a Custom Adapter [page 18]
- Debugging a Custom Adapter [page 140]

### 2.5.1 Formatters Currently Available from SAP

The adapter framework provides numerous formatter modules that come standard with smart data streaming. You can reuse these modules to create a custom adapter instance.

The Input Datatype column specifies the format of the incoming data, while the Output Datatype column specifies the format that the formatter translates this incoming data into.

AdapterRow is a container for one or more records or rows as they flow from one module (transporter, formatter, or the smart data streaming connector) to the next. You can add multiple records as objects within a List of a single AdapterRow object. The AdapterRow has a timestamp and block flags that control how its records are communicated to and from smart data streaming. See [Event Blocks](#) in the SAP HANA Smart Data Streaming: CCL Reference for additional details.

AepRecord is the class that represents stream records. This class is the type that the smart data streaming publisher expects as the data member in the AdapterRow instances that it receives from the previous module. This class is also the type used in the AdapterRow instances that the smart data streaming subscriber passes on to the next module. The default operation code (opcode) for AepRecord is INSERT. Change this opcode based on your needs.
Several of the formatters come with example data:

- CSV to ESP Formatter – see $STREAMING_HOME/adapters/framework/instances/file_csv_input/data/input.csv
- JSON String to ESP Formatter – see $STREAMING_HOME/adapters/framework/instances/file_json_input/data/article_1.json
- XML String to ESP Formatter – see $STREAMING_HOME/adapters/framework/instances/file_xmlstring_input/data/input.xml
- XMLDoc Stream to ESP Formatter – see $STREAMING_HOME/adapters/framework/instances/file_xmldoc_input/datetimeExample/data/data_1.xml
- String List to ESP Formatter – see $STREAMING_HOME/adapters/framework/examples/stringlist_input and $STREAMING_HOME/adapters/framework/examples/stringlist_output

Table 4:

<table>
<thead>
<tr>
<th>Name</th>
<th>Input Datatype</th>
<th>Mode (Streaming/Row)</th>
<th>Output Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avro Record to ESP Formatter</td>
<td>AvroRecord</td>
<td>Row</td>
<td>AepRecord</td>
<td>Translates Avro record objects to AepRecord objects.</td>
</tr>
<tr>
<td>CSV to ESP Formatter</td>
<td>Java.lang.String</td>
<td>Row</td>
<td>AepRecord</td>
<td>Translates CSV string data to AepRecord objects.</td>
</tr>
<tr>
<td>ESP to Avro Record Formatter</td>
<td>AepRecord</td>
<td>Row</td>
<td>AvroRecord</td>
<td>Converts AepRecord objects to Avro record objects.</td>
</tr>
<tr>
<td>ESP to CSV Formatter</td>
<td>AepRecord</td>
<td>Row</td>
<td>Java.lang.String</td>
<td>Translates AepRecord objects to CSV string data.</td>
</tr>
<tr>
<td>ESP to JSON String Formatter</td>
<td>AepRecord</td>
<td>Row</td>
<td>Java.lang.String</td>
<td>Translates AepRecord objects to JSON string data.</td>
</tr>
<tr>
<td>ESP to Map Formatter</td>
<td>AepRecord</td>
<td>Row</td>
<td>Java.lang.String</td>
<td>Converts AepRecord objects into Map message string data.</td>
</tr>
<tr>
<td>ESP to Object List Formatter</td>
<td>AepRecord</td>
<td>Row</td>
<td>Java.util.List&lt;Jav a.lang.Object&gt;</td>
<td>Converts AepRecord objects to Java object list.</td>
</tr>
<tr>
<td>ESP to String List Formatter</td>
<td>AepRecord</td>
<td>Row</td>
<td>Java.util.List&lt;Jav a.lang.String&gt;</td>
<td>Converts AepRecord objects to string list.</td>
</tr>
<tr>
<td>ESP to XML String Formatter</td>
<td>AepRecord</td>
<td>Row</td>
<td>Java.lang.String</td>
<td>Translates AepRecord objects to ESP XML string.</td>
</tr>
<tr>
<td>Name</td>
<td>Input Datatype</td>
<td>Mode (Streaming/Row)</td>
<td>Output Datatype</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------------------</td>
<td>----------------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ESP to XMLDoc String Formatter</td>
<td>AepRecord</td>
<td>Row</td>
<td>String</td>
<td>Translates AepRecord objects to XML format string according to the schema file configured in the adapter configuration file.</td>
</tr>
<tr>
<td>JSON String to ESP Formatter</td>
<td>Java.lang.String</td>
<td>Row</td>
<td>AepRecord</td>
<td>Translates JSON string to AepRecord objects.</td>
</tr>
<tr>
<td>JSON Stream to JSON String Formatter</td>
<td>InputStream</td>
<td>Streaming</td>
<td>Java.lang.String</td>
<td>Splits ByteStream data into standalone JSON message string data.</td>
</tr>
<tr>
<td>Map to ESP Formatter</td>
<td>Java.lang.String</td>
<td>Row</td>
<td>AepRecord</td>
<td>Converts Map message string data into AepRecord objects.</td>
</tr>
<tr>
<td>Object List to ESP Formatter</td>
<td>Java.util.List&lt;Java.lang.Object&gt;</td>
<td>Row</td>
<td>AepRecord</td>
<td>Converts Java object list to AepRecord objects.</td>
</tr>
<tr>
<td>Stream to String Formatter</td>
<td>InputStream</td>
<td>Streaming</td>
<td>Java.lang.String</td>
<td>Splits ByteStream data into strings according to the value specified in the delimiter property.</td>
</tr>
<tr>
<td>String to Stream Formatter</td>
<td>Java.lang.String</td>
<td>Streaming</td>
<td>OutputStream</td>
<td>Merges strings into ByteStream data.</td>
</tr>
<tr>
<td>String List to ESP Formatter</td>
<td>Java.util.List&lt;Java.lang.String&gt;</td>
<td>Row</td>
<td>AepRecord</td>
<td>Converts string list to AepRecord objects.</td>
</tr>
<tr>
<td>XML String to ESP Formatter</td>
<td>Java.lang.String</td>
<td>Row</td>
<td>AepRecord</td>
<td>Translates ESP XML strings to AepRecord objects.</td>
</tr>
<tr>
<td>XMLDoc Stream to ESP Formatter</td>
<td>InputStream</td>
<td>Streaming</td>
<td>AepRecord</td>
<td>Parses XML strings, extracts data according to the schema file configured in the adapter configuration file, and translates the data to AepRecord objects.</td>
</tr>
</tbody>
</table>

In this section:

Avro Record to ESP Formatter Module Properties [page 64]
The Avro Record to ESP formatter converts an Avro record object to a row for smart data streaming. Set values for this formatter in the adapter configuration file.

CSV String to ESP Formatter Module Properties [page 65]
The CSV String to ESP formatter translates CSV strings to AepRecord objects. Set values for this formatter in the adapter configuration file.

ESP to Avro Record Formatter Module Properties [page 67]
The ESP to Avro Record formatter converts a row from smart data streaming to an Avro record object. Set values for this formatter in the adapter configuration file.

ESP to CSV String Formatter Module Properties [page 68]
The ESP to CSV String formatter translates AepRecord objects to CSV strings. Set values for this formatter in the adapter configuration file.

ESP to JSON String Formatter Module Properties [page 71]
The ESP to JSON String formatter translates AepRecord objects to JSON strings, and sends the JSON strings to next streaming output transporter that is configured in the adapter configuration file. Set values for this formatter in the adapter configuration file.

ESP to Map Formatter Module Properties [page 73]
The ESP to Map formatter converts a row from smart data streaming to a map message string object. Set values for this formatter in the adapter configuration file.

ESP to Object List Formatter Module Properties [page 75]
The ESP to Object List formatter converts a row from smart data streaming to an object list. Set values for this formatter in the adapter configuration file.

ESP to String List Formatter Module Properties [page 76]
The ESP to String List formatter converts a row from smart data streaming to a string list. Set values for this formatter in the adapter configuration file.

ESP to XML String Formatter Module Properties [page 78]
The ESP to XML String formatter translates AepRecord objects to ESP XML string. Set values for this formatter in the adapter configuration file.

ESP to XMLDOC String Formatter Module Properties [page 80]
The ESP to XMLDOC String formatter translates AepRecord objects to XML format string according to the schema file specified in the adapter configuration file. Set values for this formatter in the adapter configuration file.

JSON String to ESP Formatter Module Properties [page 81]
The JSON String to ESP formatter translates JSON strings to AepRecord objects. Set values for this formatter in the adapter configuration file.

JSON Stream to JSON String Formatter Module Properties [page 85]
The JSON Stream to JSON String formatter reads data from InputStream, splits it into standalone JSON message strings, and sends these message strings to the next module that is configured in the adapter configuration file. Set values for this formatter in the adapter configuration file.

Object List to ESP Formatter Module Properties [page 85]
The Object List to ESP formatter converts an object list to a row for smart data streaming. Set values for this formatter in the adapter configuration file.

Stream to String Formatter Module Properties [page 87]
The Stream to String formatter reads streaming data from an input stream, and splits it into Java strings. Set values for this formatter in the adapter configuration file.
String to Stream Formatter Module Properties [page 88]
The String to Stream formatter writes Java strings to output streams. Set values for this formatter in the adapter configuration file.

String List to ESP Formatter Module Properties [page 89]
The String List to ESP formatter converts a string list to a row for smart data streaming. Set values for this formatter in the adapter configuration file.

XML String to ESP Formatter Module Properties [page 90]
The XML String to ESP formatter translates ESP XML strings to AepRecord objects. Set values for this formatter in the adapter configuration file.

XMLDoc Stream to ESP Formatter Module Properties [page 92]
The XMLDoc Stream to ESP formatter parses XML format strings, extracts data according to the schema file specified in the adapter configuration file, and translates this data to AepRecord objects. Set values for this formatter in the adapter configuration file.

Map to ESP Formatter Module Properties [page 95]
The Map to ESP formatter converts a map message string object to a row for smart data streaming. Set values for this formatter in the adapter configuration file.

Related Information

Accessing Adapter Toolkit API Reference Information [page 19]
Building a Custom Formatter Module [page 98]

2.5.1.1 Avro Record to ESP Formatter Module Properties

The Avro Record to ESP formatter converts an Avro record object to a row for smart data streaming. Set values for this formatter in the adapter configuration file.

This formatter is row-based and can connect two row-based transporters rather than stream-based transporters.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| AvroJsonSchemaFile     | Type: string  
(Required) The filepath to the Avro schema file for the adapter. An example schema file is installed with smart data streaming under STREAMING_HOME/adapters/framework/instances/kafka_avro_input/avroformat.json |
| SecondDateFormat       | Type: string  
(Optional) Format string for parsing SecondDate values. The default is yyyy-MM-dd'T'HH:mm:ss. |
### XML Element

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
</table>
| MsDateFormat | Type: string  
  (Advanced) The format for parsing MsDate values. For example, yyyy-MM-dd'T'HH:mm:ss.SSS is the default value. |
| TimeFormat | Type: string  
  (Advanced) The format for parsing time values. The default value is HH:mm:ss. |
| BigdatetimeFormat | Type: string  
  (Advanced) The format for parsing bigdatetime values. For example, yyyy-MM-dd'T'HH:mm:ss.SSSSSS is the default value. Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior. |

### 2.5.1.2 CSV String to ESP Formatter Module Properties

The CSV String to ESP formatter translates CSV strings to AepRecord objects. Set values for this formatter in the adapter configuration file.

This formatter is row-based and can connect two row-based transporters rather than stream-based transporters.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| ExpectStreamNameOpcode | Type: boolean  
  (Required) If set to true, the adapter interprets the first two fields of the incoming CSV line as stream name and opcode, respectively. The adapter discards messages that contain unmatched stream names.  
  The accepted opcodes are:  
  - i or I: INSERT  
  - d or D: DELETE  
  - u or U: UPDATE  
  - p or P: UPSERT  
  - s or S: SAFEDELETE  
  The default value is false. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| Delimiter            | Type: `string`  
(Advanced) Specifies the symbol or symbols used to separate the columns.  
Characters from the ASCII printable character set are supported as delimiters. The default value is a comma (,).  
Certain control characters can be specified as delimiters using the following escape codes:  
- \t – horizontal tab  
- \n – line feed  
- \r – carriage return  
- \\ – backslash character  
- \0 – NUL or null character |
| HasHeader            | Type: `boolean`  
(Advanced) Indicates whether the first line of the file contains field descriptions. Default value is false. |
| SecondDateFormat     | Type: `string`  
(Advanced) The format string for parsing SecondDate values.  
The default value is `yyyy-MM-dd'T'HH:mm:ss`. |
| MsDateFormat         | Type: `string`  
(Advanced) Format string for parsing MsDate values.  
The default value is `yyyy-MM-dd'T'HH:mm:ss.SSS`. |
| TimeFormat           | Type: `string`  
(Advanced) Specifies the format for parsing time values.  
The default value is `HH:mm:ss`. |
| BigdatetimeFormat    | Type: `string`  
(Advanced) Format string for parsing bigdatetime values.  
The default value is `yyyy-MM-dd'T'HH:mm:ss.SSSSSS`.  
Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior. |
### 2.5.1.3 ESP to Avro Record Formatter Module Properties

The ESP to Avro Record formatter converts a row from smart data streaming to an Avro record object. Set values for this formatter in the adapter configuration file.

This formatter is row-based and can connect two row-based transporters rather than stream-based transporters.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| Instances     | Type: string  
(Optional) The number of threads in which a formatter is running. The default value is 1. If the value is 0 or not specified, the formatter runs in a single thread. If the value is greater than 1, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify. |
| PreserveOrder | Type: boolean  
(Dependent required) Preserves the input row sequence formatted for output data when multi-thread mode is enabled. The default is true. If set to true, and Instances is set to greater than 1, the input and output row sequences are identical. If set to false, the input and output row sequences may be inconsistent. If Instances is set to less than 1, PreserveOrder has no effect. When guaranteed delivery (GD) and multi-thread mode are both enabled, PreserveOrder must be true to ensure that GD commits are in ascending order. |

**Note**
Parallel enables the formatter module to run as a separated thread, which is required with multi-thread instances. If you set Instances to a value greater than 1, but set Parallel to false, Parallel automatically resets to true.
### XML Element Description

**AvroJsonSchemaFile**

- **Type:** string
- **(Required)** The filepath to the Avro schema file for the adapter. An example schema file is installed with smart data streaming under `STREAMING_HOME/adapters/framework/instances/kafka_avro_input/avroformat.json`.

**SecondDateFormat**

- **Type:** string
- **(Optional)** Format string for parsing SecondDate values. The default is `yyyy-MM-dd'T'HH:mm:ss`.

**MsDateFormat**

- **Type:** string
- **(Advanced)** The format for parsing MsDate values. For example, `yyyy-MM-dd'T'HH:mm:ss.SSS` is the default value.

**TimeFormat**

- **Type:** string
- **(Advanced)** The format for parsing time values. The default value is `HH:mm:ss`.

**BigdatetimeFormat**

- **Type:** string
- **(Advanced)** The format for parsing bigdatetime values. For example, `yyyy-MM-dd'T'HH:mm:ss.SSSSSS` is the default value. Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior.

---

### 2.5.1.4 ESP to CSV String Formatter Module Properties

The ESP to CSV String formatter translates AepRecord objects to CSV strings. Set values for this formatter in the adapter configuration file.

This formatter is row-based and can connect two row-based transporters rather than streaming-based transporters.
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| PrependStreamNameOpcode     | Type: boolean  
(Optional) If set to true, the adapter prepends the stream name and the opcode in each row of generated data. The default value is false. |
| Delimiter                   | Type: string  
(Advanced) Specifies the symbol or symbols used to separate the columns.  
Characters from the ASCII printable character set are supported as delimiters. The default value is a comma (,).  
Certain control characters can be specified as delimiters using the following escape codes:  
- \t – horizontal tab  
- \n – line feed  
- \r – carriage return  
- \\ – backslash character  
- \0 – NUL or null character |
| HasHeader                   | Type: boolean  
(Advanced) Indicates whether the first line of the file contains the description of the fields (smart data streaming stream columns). If set to true, the adapter outputs the CSV header line first, then the first data record, the second data record, and so on.  
The default value is false. |
| SecondDateFormat            | Type: string  
(Advanced) The format string for SecondDate values.  
For example, yyyy-MM-dd'T'HH:mm:ss is the default value. |
| MsDateFormat                | Type: string  
(Advanced) Format string for MsDate values.  
For example, yyyy-MM-dd'T'HH:mm:ss.SSS is the default value. |
| TimeFormat                  | Type: string  
(Advanced) Specifies the format for parsing time values.  
The default value is HH:mm:ss. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigdatetimeFormat</td>
<td>Type: string (Advanced) Format string for parsing bigdatetime values. The default value is <code>yyyy-MM-dd'T'HH:mm:ss.SSSSSS</code>. Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior.</td>
</tr>
<tr>
<td>Instances</td>
<td>Type: string (Optional) The number of threads in which a formatter is running. The default value is 1. If the value is 0 or not specified, the formatter runs in a single thread. If the value is greater than 1, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> Parallel enables the formatter module to run as a separated thread, which is required with multi-thread instances. If you set <code>Instances</code> to a value greater than 1, but set <code>Parallel</code> to false, <code>Parallel</code> automatically resets to true.</td>
</tr>
<tr>
<td>PreserveOrder</td>
<td>Type: boolean (Dependent required) Preserves the input row sequence formatted for output data when multi-thread mode is enabled. The default is true. If set to true, and <code>Instances</code> is set to greater than 1, the input and output row sequences are identical. If set to false, the input and output row sequences may be inconsistent. If <code>Instances</code> is set to less than 1, <code>PreserveOrder</code> has no effect. When guaranteed delivery (GD) and multi-thread mode are both enabled, <code>PreserveOrder</code> must be true to ensure that GD commits are in ascending order.</td>
</tr>
</tbody>
</table>
2.5.1.5  ESP to JSON String Formatter Module Properties

The ESP to JSON String formatter translates AepRecord objects to JSON strings, and sends the JSON strings to next streaming output transporter that is configured in the adapter configuration file. Set values for this formatter in the adapter configuration file.

This formatter is streaming-based and can connect two stream-based transporters rather than row-based transporters.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColsMapping</td>
<td>(Required) Element containing the Column element.</td>
</tr>
<tr>
<td>Column</td>
<td>Type: string</td>
</tr>
</tbody>
</table>
|                   | (Required) A JSONPath expression for the output data. You can have multiple Column elements. Columns from the output stream or window map to these expressions in the order listed, and output as JSON strings. For example, you might have a stream with the columns name and gender. In each Column element, you would specify the JSONPath expression you want each column to map to. In this case, the entry would look like this:

```
<Column>sap.person.name</Column>
<Column>sap.person.gender</Column>
```

So, one row of the stream, with the values Peter and male, would output as the following JSON string:

```
{
    "sap": {
        "person": {
            "name": "Peter",
            "gender": "male"
        }
    }
}
```

The first column of a stream is mapped to the JSONPath expression in the first Column element, the second column of a stream is mapped to the expression in the second Column element, and so on.

No default value.

<table>
<thead>
<tr>
<th>SecondDateFormat</th>
<th>Type: string</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(Advanced) The format string for SecondDate values. For example, yyyy-MM-dd'T'HH:mm:ss is the default value.</td>
</tr>
<tr>
<td>XML Element</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MsDateFormat</td>
<td>Type: string&lt;br&gt;(Advanced) The format string for MsDate values. For example, <code>yyyy-MM-dd'T'HH:mm:ss.SSS</code> is the default value.</td>
</tr>
<tr>
<td>TimeFormat</td>
<td>Type: string&lt;br&gt;(Advanced) Specifies the format for parsing time values. The default value is <code>HH:mm:ss</code>.</td>
</tr>
<tr>
<td>BigdatetimeFormat</td>
<td>Type: string&lt;br&gt;(Advanced) Format string for parsing bigdatetime values. The default value is <code>yyyy-MM-dd'T'HH:mm:ss.SSSSSS</code>. Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior.</td>
</tr>
<tr>
<td>Instances</td>
<td>Type: string&lt;br&gt;(Optional) The number of threads in which a formatter is running. The default value is 1. If the value is 0 or not specified, the formatter runs in a single thread. If the value is greater than 1, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify.</td>
</tr>
</tbody>
</table>

**Note**

Parallel enables the formatter module to run as a separated thread, which is required with multi-thread instances. If you set Instances to a value greater than 1, but set Parallel to false, Parallel automatically resets to true.
2.5.1.6 ESP to Map Formatter Module Properties

The ESP to Map formatter converts a row from smart data streaming to a map message string object. Set values for this formatter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| PreserveOrder                    | **Type:** boolean  
(Dependent required) Preserves the input row sequence formatted for output data when multi-thread mode is enabled. The default is true.  
If set to true, and Instances is set to greater than 1, the input and output row sequences are identical. If set to false, the input and output row sequences may be inconsistent. If Instances is set to less than 1, PreserveOrder has no effect.  
When guaranteed delivery (GD) and multi-thread mode are both enabled, PreserveOrder must be true to ensure that GD commits are in ascending order. |

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| PrependStreamNameOpcode          | **Property ID:** PrependStreamNameOpcode  
**Type:** boolean  
(Optional) If set to true, the adapter inserts the stream name into the field _ESP_STREAM_NAME, and the opcode into the field _ESP_OPS. |
| ValueAsString                    | **Type:** boolean  
(Advanced) If set to true, the adapter converts all smart data streaming data type into text string. The default value is true.  
If set to false, use the CustomFieldsType parameter to define custom data type mapping. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| CustomFieldsType  | Type: string  
(Advanced) A mapping of smart data streaming datatype and external datatype. Specify the conversion of ESP data type to JMS Map datatype in the following format: `<esp-column-name1>=<custom-data-type>`. For example, `espColumn1=int`. When there are multiple pairs in a string, each pair is delimited by a colon (:``). For example, `espColumn1=int:espColumn2:long:espColumn3=string`. The column name must be case-sensitive and exist in the schema of the attached project, or the pair is ignored. Valid datatypes are `string`, `int`, `short`, `long`, `boolean`, `float`, `double`, `byte`, and `bytes`; they are case-insensitive. See Datatype Mapping for JMS Map Formatter [page 101] for a mapping conversion table listing supported ESP to JMS Map message datatypes. |
| Instances         | Type: string  
(Optional) The number of threads in which a formatter is running. The default value is 1.  
If the value is 0 or not specified, the formatter runs in a single thread. If the value is greater than 1, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify.  

**Note**  
Parallel enables the formatter module to run as a separated thread, which is required with multi-thread instances. If you set Instances to a value greater than 1, but set Parallel to false, Parallel automatically resets to true. |
| PreserveOrder     | Type: boolean  
(Dependent required) Preserves the input row sequence formatted for output data when multi-thread mode is enabled. The default is true.  
If set to true, and Instances is set to greater than 1, the input and output row sequences are identical. If set to false, the input and output row sequences may be inconsistent. If Instances is set to less than 1, PreserveOrder has no effect.  
When guaranteed delivery (GD) and multi-thread mode are both enabled, PreserveOrder must be true to ensure that GD commits are in ascending order. |
### SecondDateFormat

**Type:** string

(Advanced) The format string for SecondDate values.

For example, `yyyy-MM-dd'T'HH:mm:ss` is the default value.

### MsDateFormat

**Type:** string

(Advanced) Format string for MsDate values.

For example, `yyyy-MM-dd'T'HH:mm:ss.SSS` is the default value.

### TimeFormat

**Type:** string

(Advanced) Specifies the format for parsing time values.

The default value is `HH:mm:ss`.

### BigdatetimeFormat

**Type:** string

(Advanced) Specify the format for parsing bigdatetime values.

For example, `yyyy-MM-dd'T'HH:mm:ss.SSSSSS` is the default value. Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior.

---

### 2.5.1.7 ESP to Object List Formatter Module Properties

The ESP to Object List formatter converts a row from smart data streaming to an object list. Set values for this formatter in the adapter configuration file.

#### XML Element

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| SecondDateFormat | Type: string  
(Advanced) The format string for SecondDate values. For example, `yyyy-MM-dd'T'HH:mm:ss` is the default value. |
| MsDateFormat | Type: string  
(Advanced) Format string for MsDate values. For example, `yyyy-MM-dd'T'HH:mm:ss.SSS` is the default value. |
| TimeFormat | Type: string  
(Advanced) Specifies the format for parsing time values. The default value is `HH:mm:ss`. |
| BigdatetimeFormat | Type: string  
(Advanced) Specify the format for parsing bigdatetime values. For example, `yyyy-MM-dd'T'HH:mm:ss.SSSSSS` is the default value. Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior. |
| OutputAsSQLDatetimeFormat | Type: boolean  
(Optional) Indicates whether the smart data streaming `seconddate`, `msdate`, and `bigdatetime` datatypes are output as `java.sql.Date` or `java.sql.Timestamp`. The default output is `java.util.Date`. |
### ESP to String List Formatter Module Properties

The ESP to String List formatter converts a row from smart data streaming to a string list. Set values for this formatter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| Instances         | Type: string  
|(Optional) The number of threads in which a formatter is running. The default value is 1.  
If the value is 0 or not specified, the formatter runs in a single thread. If the value is greater than 1, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify. |

**Note**  
Parallel enables the formatter module to run as a separated thread, which is required with multi-thread instances. If you set Instances to a value greater than 1, but set Parallel to false, Parallel automatically resets to true.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| PreserveOrder     | Type: boolean  
|(Dependent required) Preserves the input row sequence formatted for output data when multi-thread mode is enabled. The default is true.  
If set to true, and Instances is set to greater than 1, the input and output row sequences are identical. If set to false, the input and output row sequences may be inconsistent. If Instances is set to less than 1, PreserveOrder has no effect.  
When guaranteed delivery (GD) and multi-thread mode are both enabled, PreserveOrder must be true to ensure that GD commits are in ascending order. |

The default is `yyyy-MM-dd'T'HH:mm:ss`.  

- **2.5.1.8 ESP to String List Formatter Module Properties**
- **XML Element**
- **Description**
- **Instances**
- **Type**: string  
| (Optional) The number of threads in which a formatter is running. The default value is 1.  
If the value is 0 or not specified, the formatter runs in a single thread. If the value is greater than 1, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify. |

**Note**  
Parallel enables the formatter module to run as a separated thread, which is required with multi-thread instances. If you set Instances to a value greater than 1, but set Parallel to false, Parallel automatically resets to true.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| PreserveOrder     | Type: boolean  
|(Dependent required) Preserves the input row sequence formatted for output data when multi-thread mode is enabled. The default is true.  
If set to true, and Instances is set to greater than 1, the input and output row sequences are identical. If set to false, the input and output row sequences may be inconsistent. If Instances is set to less than 1, PreserveOrder has no effect.  
When guaranteed delivery (GD) and multi-thread mode are both enabled, PreserveOrder must be true to ensure that GD commits are in ascending order. |

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| SecondDateFormat  | Type: string  
|(Optional) Format string for parsing SecondDate values.  
The default is `yyyy-MM-dd'T'HH:mm:ss`. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| MsDateFormat        | **Type:** string  
                        (Optional) Format string for parsing MsDate values. 
                        The default is `yyyy-MM-dd'T'HH:mm:ss.SSS`.                                                                                                      |
| TimeFormat          | **Type:** string  
                        (Advanced) Specifies the format for parsing time values.  
                        The default value is `HH:mm:ss`.                                                                                                                 |
| BigdatetimeFormat   | **Type:** string  
                        (Advanced) Format string for parsing bigdatetime values.  
                        The default value is `yyyy-MM-dd'T'HH:mm:ss.SSSSSS`.                                                                                           
                        Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior. |
| Instances           | **Type:** string  
                        (Optional) The number of threads in which a formatter is running. The default value is 1.  
                        If the value is 0 or not specified, the formatter runs in a single thread. If the value is greater than 1, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify. |

**Note**

`Parallel` enables the formatter module to run as a separated thread, which is required with multi-thread instances. If you set `Instances` to a value greater than 1, but set `Parallel` to false, `Parallel` automatically resets to true.
### PreserveOrder

**Type:** boolean

(Required) Preserves the input row sequence formatted for output data when multi-thread mode is enabled. The default is true.

If set to true, and `Instances` is set to greater than 1, the input and output row sequences are identical. If set to false, the input and output row sequences may be inconsistent. If `Instances` is set to less than 1, `PreserveOrder` has no effect.

When guaranteed delivery (GD) and multi-thread mode are both enabled, `PreserveOrder` must be true to ensure that GD commits are in ascending order.

### 2.5.1.9 ESP to XML String Formatter Module Properties

The ESP to XML String formatter translates AepRecord objects to ESP XML string. Set values for this formatter in the adapter configuration file.

This formatter is row-based and can connect two row-based transporters rather than stream-based transporters.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| **SecondDateFormat** | Type: string  
(Optional) Format string for parsing SecondDate values.  
The default is `yyyy-MM-dd'T'HH:mm:ss`. |
| **MsDateFormat**     | Type: string  
(Optional) Format string for parsing MsDate values.  
The default is `yyyy-MM-dd'T'HH:mm:ss.SSS`. |
| **TimeFormat**       | Type: string  
(Advanced) Specifies the format for parsing time values.  
The default value is `HH:mm:ss`. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| BigdatetimeFormat   | Type: string  
(Advanced) Specify the format for parsing bigdatetime values.  
For example, yyyy-MM-dd'T'HH:mm:ss.SSSSSS is the default value. Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior. |
| Instances           | Type: string  
(Optional) The number of threads in which a formatter is running. The default value is 1.  
If the value is 0 or not specified, the formatter runs in a single thread. If the value is greater than 1, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify. |
| PreserveOrder       | Type: boolean  
(Dependent required) Preserves the input row sequence formatted for output data when multi-thread mode is enabled. The default is true.  
If set to true, and Instances is set to greater than 1, the input and output row sequences are identical. If set to false, the input and output row sequences may be inconsistent. If Instances is set to less than 1, PreserveOrder has no effect.  
When guaranteed delivery (GD) and multi-thread mode are both enabled, PreserveOrder must be true to ensure that GD commits are in ascending order. |
2.5.1.10 ESP to XMLDOC String Formatter Module Properties

The ESP to XMLDOC String formatter translates AepRecord objects to XML format string according to the schema file specified in the adapter configuration file. Set values for this formatter in the adapter configuration file.

This formatter is stream-based and can connect two stream-based transporters rather than row-based transporters.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| XMLSchemaFilePath           | Type: string
(Required) The path to the XML schema file that the XML output document builds against. No default value. |
| GlobalElementLocalName      | Type: string
(Required) A global element to use as the root element in the generated XML document. No default value. |
| ColsMapping                 | (Required) Element containing the Column element.                           |
| Column                      | Type: string
(Required) The attributes or child elements, generated by the global element that match by a pattern path expression and map to columns of the smart data streaming row. For example, [Column.XPath expression</Column>]+. The XPath expression is any valid XPath expression specified by an XPath specification. The first Column/> element value is mapped to the first column of the smart data streaming row, the second Column/> element value is mapped to the second column of the smart data streaming row, and so on. |
### XML Element | Description
--- | ---
**Instances** | Type: string  
(Optional) The number of threads in which a formatter is running. The default value is 1.  
If the value is 0 or not specified, the formatter runs in a single thread. If the value is greater than 1, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify.

**Note**
Parallel enables the formatter module to run as a separated thread, which is required with multi-thread instances. If you set `Instances` to a value greater than 1, but set `Parallel` to false, `Parallel` automatically resets to true.

**PreserveOrder** | Type: boolean  
(Dependent required) Preserves the input row sequence formatted for output data when multi-thread mode is enabled. The default is true.  
If set to true, and `Instances` is set to greater than 1, the input and output row sequences are identical. If set to false, the input and output row sequences may be inconsistent. If `Instances` is set to less than 1, `PreserveOrder` has no effect.  
When guaranteed delivery (GD) and multi-thread mode are both enabled, `PreserveOrder` must be true to ensure that GD commits are in ascending order.

## 2.5.1.11 JSON String to ESP Formatter Module Properties

The JSON String to ESP formatter translates JSON strings to AepRecord objects. Set values for this formatter in the adapter configuration file.

This formatter is row-based and can connect two row-based transporters rather than streaming-based transporters.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ColumnMappings</strong></td>
<td>(Required) Element containing the <code>ColsMapping</code> element.</td>
</tr>
<tr>
<td>XML Element</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| ColsMapping  | Type: complexType  
(Required) Element that contains the Column element. You can have multiple ColsMapping elements if you are using an ESPMultiStreamPublisher.  
This element has two attributes:  
**streamname**  
The smart data streaming stream to which you want to publish JSON data.  
**rootpath**  
A root path for the JSON data. Must be an array. See the Column element for an example of a root path value. |
**XML Element** | **Description**
--- | ---
Column | Type: string

(Required) A JSONPath expression for the JSON data that you want to map to columns of a stream. This expression is matched to the value specified in the rootpath attribute of the ColsMapping element, if applicable. You can have multiple Column elements.

There are two types of JSON data: array or object. For example, if you had the following JSON data about a person, you could get the individual’s first name by using the JSONPath expression firstname:

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "Fax",
      "number": "646 555-4567"
    }
  ],
  "friends": [
    ["female1","female2","female3"],
    ["male1","male2","male3"]
  ]
}
```

If you want the first phone number, specify phoneNumbers[0].number as the JSONPath expression. You would not have to specify a rootpath value.

If you want the numbers and types of phone numbers, specify phoneNumbers as the rootpath value and numbers and type as the JSONPath expressions in the Column element.

You can also specify * for Column that indicates you want all the data in an array (that does not have key value data within it). For example, if you wanted all the female friends data, specify friends[1] for rootpath and * for Column.

The first <Column/> element and its value are mapped to the first column of the stream, the second <Column/> element and its value are mapped to the second column of the stream, and so on.
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecondDateFormat</td>
<td>Type: string</td>
</tr>
<tr>
<td></td>
<td>(Advanced) The format string for parsing SecondDate values.</td>
</tr>
<tr>
<td></td>
<td>For example, yyyy-MM-dd'T'HH:mm:ss is the default value.</td>
</tr>
<tr>
<td>MsDateFormat</td>
<td>Type: string</td>
</tr>
<tr>
<td></td>
<td>(Advanced) The format string for parsing MsDate values.</td>
</tr>
<tr>
<td></td>
<td>For example, yyyy-MM-dd'T'HH:mm:ss.SSS is the default value.</td>
</tr>
<tr>
<td>TimeFormat</td>
<td>Type: string</td>
</tr>
<tr>
<td></td>
<td>(Advanced) Specifies the format for parsing time values.</td>
</tr>
<tr>
<td></td>
<td>The default value is HH:mm:ss.</td>
</tr>
<tr>
<td>BigdatetimeFormat</td>
<td>Type: string</td>
</tr>
<tr>
<td></td>
<td>(Advanced) Format string for parsing bigdatetime values.</td>
</tr>
<tr>
<td></td>
<td>The default value is yyyy-MM-dd'T'HH:mm:ss.SSSSSS.</td>
</tr>
<tr>
<td></td>
<td>Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior.</td>
</tr>
<tr>
<td>Instances</td>
<td>Type: string</td>
</tr>
<tr>
<td></td>
<td>(Optional) The number of threads in which a formatter is running. The default value is 1.</td>
</tr>
<tr>
<td></td>
<td>If the value is 0 or not specified, the formatter runs in a single thread. If the value is greater than 1, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>Parallel enables the formatter module to run as a separated thread, which is required with multi-thread instances. If you set Instances to a value greater than 1, but set Parallel to false, Parallel automatically resets to true.</td>
</tr>
</tbody>
</table>

SAP HANA Smart Data Streaming: Building Custom Adapters

Smart Data Streaming Adapter Toolkit
### 2.5.1.12 JSON Stream to JSON String Formatter Module Properties

The JSON Stream to JSON String formatter reads data from InputStream, splits it into standalone JSON message strings, and sends these message strings to the next module that is configured in the adapter configuration file. Set values for this formatter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CharsetName</td>
<td>Type: string (Optional) The name of a supported charset. The default value is US-ASCII.</td>
</tr>
</tbody>
</table>

### 2.5.1.13 Object List to ESP Formatter Module Properties

The Object List to ESP formatter converts an object list to a row for smart data streaming. Set values for this formatter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecondDateFormat</td>
<td>Type: string (Optional) Format string for parsing SecondDate values. The default is yyyy-MM-dd'T'HH:mm:ss.</td>
</tr>
<tr>
<td>XML Element</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| MsDateFormat           | **Type:** string  
                        **(Optional) Format string for parsing MsDate values.**  
                        The default is `yyyy-MM-dd'T'HH:mm:ss.SSS`.                                                                                              |
| TimeFormat             | **Type:** string  
                        **(Advanced) Specifies the format for parsing time values.**  
                        The default value is `HH:mm:ss`.                                                                                                         |
| BigdatetimeFormat      | **Type:** string  
                        **(Advanced) Specify the format for parsing bigdatetime values.**  
                        For example, `yyyy-MM-dd'T'HH:mm:ss.SSSSSS` is the default value. Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior. |
| Instances              | **Type:** string  
                        **(Optional) The number of threads in which a formatter is running.** The default value is 1.  
                        If the value is 0 or not specified, the formatter runs in a single thread. If the value is greater than 1, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify. |

**Note**

Parallel enables the formatter module to run as a separated thread, which is required with multi-thread instances. If you set **Instances** to a value greater than 1, but set **Parallel** to false, **Parallel** automatically resets to true.
### 2.5.1.14 Stream to String Formatter Module Properties

The Stream to String formatter reads streaming data from an input stream, and splits it into Java strings. Set values for this formatter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Delimiter**      | Type: string  
(Required) The symbol or symbols that separate columns. Use only character delimiters from the ASCII printable character set. The default value is "\n". |
| **IncludeDelimiter** | Type: boolean  
(Required) If set to true, the delimiter is part of current row. If set to false, the delimiter is not part of the current row. The default value is false. |
| **AppendString**   | Type: string  
(Required if IncludeDelimiter is set to true) If set to true, specify the string to append to the end of each result row. No default value. |
| **AppendPosition** | Type: string  
(Required if IncludeDelimiter is set to true) The position at which the AppendString element takes effect. Valid values are front and end. The default value is front. |
### 2.5.1.15 String to Stream Formatter Module Properties

The String to Stream formatter writes Java strings to output streams. Set values for this formatter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| IgnoreSpace      | Type: boolean  
(Required) Indicates whether to trim the space character. The default value is true. |
| CharsetName      | Type: string  
(Advanced) The name of a supported character set. The default value is US-ASCII. |
| Delimiter        | Type: string  
(Required) The symbol or symbols that separate columns. Use only character delimiters from the ASCII printable character set. The default value is " \n ". |
| IncludeDelimiter | Type: boolean  
(Required) If set to true, the delimiter is part of current row. If set to false, the delimiter is not part of the current row. The default value is false. |
| AppendString     | Type: string  
(Required if IncludeDelimiter is set to true) If set to true, specify the string to append to the end of each result row. No default value. |
| AppendPosition   | Type: string  
(Required if IncludeDelimiter is set to true) The position to which the AppendString element takes effect. The valid values are front and end. The default value is front. |
| IgnoreSpace      | Type: boolean  
(Required) Indicates whether to ignore the space character. The default value is false. |
### 2.5.1.16 String List to ESP Formatter Module Properties

The String List to ESP formatter converts a string list to a row for smart data streaming. Set values for this formatter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CharSetName</td>
<td>Type: string&lt;br&gt;(Advanced) The name of a supported character set. The default value is US-ASCII.</td>
</tr>
<tr>
<td>SecondDateFormat</td>
<td>Type: string&lt;br&gt;(Optional) Format string for parsing SecondDate values. The default is <code>yyyy-MM-dd'T'HH:mm:ss</code>.</td>
</tr>
<tr>
<td>MsDateFormat</td>
<td>Type: string&lt;br&gt;(Optional) Format string for parsing MsDate values. The default is <code>yyyy-MM-dd'T'HH:mm:ss.SSS</code>.</td>
</tr>
<tr>
<td>TimeFormat</td>
<td>Type: string&lt;br&gt;(Advanced) Specifies the format for parsing time values. The default value is <code>HH:mm:ss</code>.</td>
</tr>
<tr>
<td>BigdatetimeFormat</td>
<td>Type: string&lt;br&gt;(Advanced) Format string for parsing bigdatetime values. The default value is <code>yyyy-MM-dd'T'HH:mm:ss.SSSSSS</code>. &lt;br&gt;Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior.</td>
</tr>
</tbody>
</table>
### XML Element: Instances

**Type:** string

(Optional) The number of threads in which a formatter is running. The default value is 1.

If the value is 0 or not specified, the formatter runs in a single thread. If the value is greater than 1, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify.

**Note**

Parallel enables the formatter module to run as a separated thread, which is required with multi-thread instances. If you set `Instances` to a value greater than 1, but set `Parallel` to false, `Parallel` automatically resets to true.

### XML Element: PreserveOrder

**Type:** boolean

(Dependent required) Preserves the input row sequence formatted for output data when multi-thread mode is enabled. The default is true.

If set to true, and `Instances` is set to greater than 1, the input and output row sequences are identical. If set to false, the input and output row sequences may be inconsistent. If `Instances` is set to less than 1, `PreserveOrder` has no effect.

When guaranteed delivery (GD) and multi-thread mode are both enabled, `PreserveOrder` must be true to ensure that GD commits are in ascending order.

### 2.5.1.17 XML String to ESP Formatter Module Properties

The XML String to ESP formatter translates ESP XML strings to AepRecord objects. Set values for this formatter in the adapter configuration file.

This formatter is row-based and can connect two row-based transporters rather than streaming-based transporters.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| SecondDateFormat | Type: string

(Optional) Format string for parsing SecondDate values. The default is `yyyy-MM-dd'T'HH:mm:ss`.

---

SAP HANA Smart Data Streaming: Building Custom Adapters
Smart Data Streaming Adapter Toolkit
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| MsDateFormat      | Type: string  
(Optional) Format string for parsing MsDate values.  
The default is `yyyy-MM-dd'T'HH:mm:ss.SSS`.                                                                                               |
| TimeFormat        | Type: string  
(Advanced) Specifies the format for parsing time values.  
The default value is `HH:mm:ss`.                                                                                                                  |
| BigdatetimeFormat | Type: string  
(Advanced) Specify the format for parsing bigdatetime values.  
For example, `yyyy-MM-dd'T'HH:mm:ss.SSSSSS` is the default value. Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior. |
| Instances         | Type: string  
(Optional) The number of threads in which a formatter is running. The default value is `1`.  
If the value is `0` or not specified, the formatter runs in a single thread. If the value is greater than `1`, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify. |

**Note**

`Parallel` enables the formatter module to run as a separated thread, which is required with multi-thread instances. If you set `Instances` to a value greater than `1`, but set `Parallel` to `false`, `Parallel` automatically resets to `true`. 
### 2.5.1.18 XMLDoc Stream to ESP Formatter Module Properties

The XMLDoc Stream to ESP formatter parses XML format strings, extracts data according to the schema file specified in the adapter configuration file, and translates this data to AepRecord objects. Set values for this formatter in the adapter configuration file.

This formatter is streaming-based and can connect two streaming-based transporters rather than row-based transporters.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreserveOrder</td>
<td>Type: boolean</td>
</tr>
<tr>
<td></td>
<td>(Dependent required) Preserves the input row sequence formatted for output data when multi-thread mode is enabled. The default is true.</td>
</tr>
<tr>
<td></td>
<td>If set to true, and Instances is set to greater than 1, the input and output row sequences are identical. If set to false, the input and output row sequences may be inconsistent. If Instances is set to less than 1, PreserveOrder has no effect.</td>
</tr>
<tr>
<td></td>
<td>When guaranteed delivery (GD) and multi-thread mode are both enabled, PreserveOrder must be true to ensure that GD commits are in ascending order.</td>
</tr>
<tr>
<td>XML Element</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| XmlElemMappingRowPattern       | Type: string (Required) A pattern to indicate which XML elements in the XML doc are processed by the formatter. The matched elements are mapped to smart data streaming rows with attributes and child elements are mapped as columns of a row. The adapter ignores any XML elements that do not match this pattern.  
This pattern is a subset of XPath expressions. The [/]*NCName[/NCName]* path expression is the only supported expression, where NCName (Non-Colonized Name) is the local name element without a prefix or namespace.  
If the elements in the path expression include a namespace URI (prefix), all of these elements belong to the same namespace. Provide the namespace in the XmlElemNamespace element.  
Here are some examples of valid path expressions:  
- /RootElement  
- ParentElement  
- ParentElement/ChildElement  
- /RootElement/ParentElement |
| XmlElemNamespace               | Type: string (Required) The namespace URI for elements that appear in the pattern path expression.                                                                                                                                                                                                                                           |
| ColsMapping                    | (Required) Element containing the Column element.                                                                                                                                                                                                                                                                                              |
| Column                         | Type: string (Required) The attributes or child elements of the XML elements that match by pattern path expression and map to columns of the smart data streaming row. For example, [<Column>XPath expression</Column>]*.  
The XPath expression is any valid XPath expression specified by an XPath specification. The XPath expression can begin only from the last XML element that appears in the path pattern expression or its decendent elements.  
The first <Column/> element value is mapped to the first column of the smart data streaming row, the second <Column/> element value is mapped to the second column of the smart data streaming row, and so on. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecondDateFormat</td>
<td>Type: string (Optional) Format string for parsing SecondDate values.</td>
</tr>
<tr>
<td></td>
<td>The default is yyyy-MM-dd'T'HH:mm:ss.</td>
</tr>
<tr>
<td>MsDateFormat</td>
<td>Type: string (Optional) Format string for parsing MsDate values.</td>
</tr>
<tr>
<td></td>
<td>The default is yyyy-MM-dd'T'HH:mm:ss.SSS.</td>
</tr>
<tr>
<td>TimeFormat</td>
<td>Type: string (Advanced) Specifies the format for parsing time values.</td>
</tr>
<tr>
<td></td>
<td>The default value is HH:mm:ss.</td>
</tr>
<tr>
<td>BigdatetimeFormat</td>
<td>Type: string (Advanced) Specify the format for parsing bigdatetime values.</td>
</tr>
<tr>
<td></td>
<td>For example, yyyy-MM-dd'T'HH:mm:ss.SSSSSS is the default value. Using less</td>
</tr>
<tr>
<td></td>
<td>than six Ss gives precision to that exact number of Ss and ignores values</td>
</tr>
<tr>
<td></td>
<td>past that specification. Using more than six Ss truncates any values beyond</td>
</tr>
<tr>
<td></td>
<td>the sixth, and replaces them with zero. This may result in slower behavior.</td>
</tr>
</tbody>
</table>
## 2.5.1.19 Map to ESP Formatter Module Properties

The Map to ESP formatter converts a map message string object to a row for smart data streaming. Set values for this formatter in the adapter configuration file.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| ExpectStreamOpcodeName | Type: boolean  
(Required) If set to true, the adapter inserts the stream name into the field `_ESP_STREAM_NAME`, and the opcode into the field `_ESP_OPS`.  
The accepted opcodes are:  
- i or I: INSERT  
- d or D: DELETE  
- u or U: UPDATE  
- p or P: UPSERT  
- s or S: SAFEDELETE  
The default value is false. |
| ValueAsString        | Type: boolean  
(Advanced) If set to true, the adapter assumes all incoming datatypes as text string and interprets them into smart data streaming data types. The default value is true. |
| Instances            | Type: string  
(Optional) The number of threads in which a formatter is running. The default value is 1.  
If the value is 0 or not specified, the formatter runs in a single thread. If the value is greater than 1, multi-thread mode is enabled, and the formatter runs in the number of threads equivalent to the value you specify. |

### Note

Parallel enables the formatter module to run as a separated thread, which is required with multi-thread instances. If you set Instances to a value greater than 1, but set Parallel to false, Parallel automatically resets to true.
### XML Element

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| **PreserveOrder** | Type: boolean  
(Dependent required) Preserves the input row sequence formatted for output data when multi-thread mode is enabled. The default is true.  
If set to true, and Instances is set to greater than 1, the input and output row sequences are identical. If set to false, the input and output row sequences may be inconsistent. If Instances is set to less than 1, PreserveOrder has no effect.  
When guaranteed delivery (GD) and multi-thread mode are both enabled, PreserveOrder must be true to ensure that GD commits are in ascending order. |
| **SecondDateFormat** | Type: string  
(Optional) Format string for parsing SecondDate values.  
The default is `yyyy-MM-dd'T'HH:mm:ss`. |
| **MsDateFormat** | Type: string  
(Optional) Format string for parsing MsDate values.  
The default is `yyyy-MM-dd'T'HH:mm:ss.SSS`. |
| **TimeFormat** | Type: string  
(Advanced) Specifies the format for parsing time values.  
The default value is `HH:mm:ss`. |
| **BigdatetimeFormat** | Type: string  
(Advanced) Specify the format for parsing bigdatetime values.  
For example, `yyyy-MM-dd'T'HH:mm:ss.SSSSSS` is the default value. Using less than six Ss gives precision to that exact number of Ss and ignores values past that specification. Using more than six Ss truncates any values beyond the sixth, and replaces them with zero. This may result in slower behavior. |
2.5.2 Datatype Mapping for Formatters

Mapping information for smart data streaming to Java datatypes and Java to smart data streaming datatypes.

ESP to Java Objects Datatype Mappings

<table>
<thead>
<tr>
<th>ESP Datatype</th>
<th>Java Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>boolean</td>
<td>java.lang.Boolean</td>
</tr>
<tr>
<td>integer</td>
<td>java.lang.Integer</td>
</tr>
<tr>
<td>long or interval</td>
<td>java.lang.Long</td>
</tr>
<tr>
<td>double</td>
<td>java.lang.Double</td>
</tr>
<tr>
<td>msdate</td>
<td>java.util.Date or java.sql.Timestamp (#OutputAsSQLDatetimesFormat is set to true)</td>
</tr>
<tr>
<td>seconndate</td>
<td>java.util.Date or java.sql.Timestamp (#OutputAsSQLDatetimesFormat is set to true)</td>
</tr>
<tr>
<td>bigdatetime</td>
<td>java.lang.String (the output format is yyyy-MM-ddTHH:mm:ss:UUUU where U stands for microseconds and the timezone is in UTC) or java.sql.Timestamp (#OutputAsSQLDatetimesFormat is set to true)</td>
</tr>
<tr>
<td>time</td>
<td>java.lang.String (the output format is HH:mm:ss) or java.sql.Time (#OutputAsSQLDatetimesFormat is set to true)</td>
</tr>
<tr>
<td>fixeddecimal</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>money,money01-money15</td>
<td>java.math.BigDecimal (it has the same precision as corresponding money(xx) object).</td>
</tr>
<tr>
<td>binary</td>
<td>byte[]</td>
</tr>
</tbody>
</table>

Java Objects to ESP Datatype Mappings

<table>
<thead>
<tr>
<th>Java Datatype</th>
<th>ESP Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java.util.Date (including its child classes)</td>
<td>bigdatetime or time</td>
</tr>
<tr>
<td>byte[]</td>
<td>binary</td>
</tr>
<tr>
<td>Java Datatype</td>
<td>ESP Datatype</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td><code>Java.lang.Object (including its child classes)</code></td>
<td>string</td>
</tr>
<tr>
<td>Any Java classes with the <code>toString()</code> method, and the result string is [true,false]</td>
<td>boolean</td>
</tr>
<tr>
<td>Any Java classes with the <code>toString()</code> method, and the result string can be used in <code>integer.valueOf(result)</code>.</td>
<td>integer</td>
</tr>
<tr>
<td>Any Java classes with the <code>toString()</code> method, and the result string can be used in <code>Long.valueOf(result)</code>.</td>
<td>long or interval</td>
</tr>
<tr>
<td>Any Java classes with the <code>toString()</code> method, and the result string can be used in <code>Double.valueOf(result)</code>.</td>
<td>double</td>
</tr>
<tr>
<td>Any Java classes with the <code>toString()</code> method. The result string consists of the decimal digits, except that the first character may be an ASCII minus sign '-' ('\u002D') to indicate a negative value, or an ASCII plus sign '+' ('\u002B') to indicate a positive value, or the period may occur in the middle of the string. The part after the period is the implied precision, and the precision matches the smart data streaming datatype.</td>
<td><code>money,money01...money15</code></td>
</tr>
<tr>
<td>Any Java classes with the <code>toString()</code> method. The result string represents a fixed decimal in the standard smart data streaming format.</td>
<td>fixeddecimal</td>
</tr>
</tbody>
</table>

### 2.5.3 Building a Custom Formatter Module

Use the smart data streaming adapter toolkit to build a custom formatter module to use within the adapter instance of your choice.

### Prerequisites

(Optional) See the `$STREAMING_HOME/adapters/framework/examples/src` directory for source code for sample formatters.
Procedure

1. Create a class that extends one of these Java classes:
   - (Row-based formatter) com.sybase.esp.adapter.framework.module.RowFormatter
   - (Streaming-based formatter) com.sybase.esp.adapter.framework.module.StreamingFormatter

   Make row-based formatters a subclass of RowFormatter, and stream-based formatters a subclass of StreamingFormatter. Use row-based formatters with row-based transporters, and stream-based formatters with stream-based transporters.

2. For row-based formatters, implement these functions:
   a. The `init()` function.
      Prepare your formatter module to convert between data formats; for example, obtain properties from the adapter configuration file and perform any required initialization tasks.
   b. The `destroy()` function.
      Perform clean-up actions for your formatter.
   c. The `convert()` function.

      Here is a simple example of a `convert()` function that converts Java objects to strings:

      ```java
      public AdapterRow convert(AdapterRow in) throws Exception {
          Object obj = in.getData(0);
          in.setData(0, obj.toString());
          return in;
      }
      ```

3. For stream-based formatters, implement these functions:
   a. The `init()` function.
      Prepare your formatter module to convert between data formats; for example, obtain properties from the adapter configuration file and perform any required initialization tasks.
   b. The `start()` function.
      Perform any necessary tasks when the adapter is started.
   c. The `execute()` function.

      Here is an example of the `execute()` function for a formatter that converts row-based data into stream-based:

      ```java
      public void execute() throws Exception {
          OutputStream output = utility.getOutputStream();
          while(!utility.isStopRequested()){
              AdapterRow row = utility.getRow();
              if(row != null){
                  AepRecord record = (AepRecord)row.getData(0);
                  String str = record.getValues().toString() + "\n";
                  output.write(str.getBytes());
              }
          }
      }
      ```

      For a formatter that converts from stream-based data into row-based, use:
      - `utility.getInputStream()` to obtain the InputStream
      - `utility.createRow()` to create the AdapterRow objects
utility.sendRow() to send the rows to the next module specified in the adapter configuration file.

d. The stop() function.
Perform any necessary tasks when the adapter is stopped.
e. The destroy() function.
Perform clean-up actions for your formatter.

4. (Optional) Call one of these functions within the functions listed in the steps above:
   ○ utility.getParameters() – to get parameters that are defined in the adapter configuration file.
   ○ utility.sendRow() – to send data to the next module defined in the adapter configuration file.
   ○ utility.getRow() – to obtain data from the previous module defined in the adapter configuration file.
   ○ utility.isStopRequested() – to determine whether a stop command has been issued.

5. Register the implemented Java class to $STREAMING_CUSTOM_ADAPTERS_HOME/config/custommodulesdefine.xml. In this example, <Parameters-Node-Name> is the optional node that represents the formatter subnode name in the adapter configuration file:

   <FormatterDefn>
   <Name>SampleFormatter</Name>
   <Class>com.sybase.esp.adapter.formatters.SampleFormatter</Class>
   <InputData>String</InputData>
   <OutputData>ESP</OutputData>
   <Parameters-Node-Name>SampleFormatterParameters</Parameters-Node-Name>
   </FormatterDefn>

6. Add the schema definitions for any unique parameters of the newly created module to the $STREAMING_CUSTOM_ADAPTERS_HOME/config/parametersdefine.xsd file.

   If any of the parameters for the newly created module are the same as parameters for the standard formatter modules, you don't need to add schema definitions for these parameters.

7. Copy the .jar file containing the class you previously implemented to $STREAMING_CUSTOM_ADAPTERS_HOME/libj.

8. (Optional) Start the adapter instance by issuing the following, where <config-file> is the adapter configuration file in which you specified the adapter instance using the newly created formatter module:
   ○ For Windows, %STREAMING_HOME%\adapters\framework\bin\start.bat <config-file>
   ○ For Unix, $STREAMING_HOME/adapters/framework/bin/start.sh <config-file>

9. (Optional) Stop the adapter instance by issuing the following, where <config-file> is the adapter configuration file in which you specified the adapter instance using the newly created formatter module:
   ○ For Windows, %STREAMING_HOME%\adapters\framework\bin\stop.bat <config-file>
   ○ For Unix, $STREAMING_HOME/adapters/framework/bin/stop.sh <config-file>

Example
See $STREAMING_HOME/adapters/framework/examples for additional details and formatter examples, as well as $STREAMING_HOME/adapters/framework/examples/src for the source code for these examples.
Next Steps

Create an adapter configuration (.xml) file to define which adapter instance uses this newly created formatter module.

Task overview: Create a Custom Adapter [page 18]

Previous task: Building a Custom Transporter Module [page 57]

Next task: Enabling Guaranteed Delivery for an Input Transporter [page 106]

Related Information

Accessing Adapter Toolkit API Reference Information [page 19]
Formatters Currently Available from SAP [page 60]

2.5.4 Datatype Mapping for JMS Map Formatter

Mapping information for smart data streaming to JMS MapMessage datatypes and JMS MapMessage to smart data streaming datatypes.

ESP to JMS MapMessage Datatype Mappings

In the following table, ESP datatypes are represented in the columns and JMS MapMessages are represented in the rows:

<table>
<thead>
<tr>
<th></th>
<th>Boolean</th>
<th>Int</th>
<th>Short</th>
<th>Long</th>
<th>Float</th>
<th>Double</th>
<th>String</th>
<th>Byte</th>
<th>Byte[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>D</td>
<td>Y*</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LONG</td>
<td>Y*</td>
<td>Y*</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERVAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOUBLE</td>
<td></td>
<td></td>
<td></td>
<td>Y*</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECOND-DATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSDATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
D=default Y=supported Y*=supported, provided the data value fits

### JMS MapMessage to ESP Datatype Mappings

In the following table, JMS MapMessages are represented in the columns and ESP datatypes are represented in the rows:

<table>
<thead>
<tr>
<th>JMS MapMessage</th>
<th>Boolean</th>
<th>Int</th>
<th>Short</th>
<th>Long</th>
<th>Float</th>
<th>Double</th>
<th>String</th>
<th>Byte</th>
<th>Byte[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>LONG</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>INTERVAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>DOUBLE</td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECOND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIGDATE-TIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STRING</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MONEY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MONEY (01-15)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIXEDDECIMAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Y=supported
2.6 Batch Processing

Details about how to control the manner in which AdapterRow instances are sent and processed by smart data streaming.

Sending individual AdapterRow instances to smart data streaming results in minimal latency, but can reduce overall throughput due to network overhead. Sending AdapterRow instances using batch processing or blocks can improve overall throughput with some reduction in latency. See Batch Processing in the SAP HANA Smart Data Streaming: Developer Guide.

When not part of a block, AdapterRow instances are published individually over the network to smart data streaming when the sendRow() or sendRowData() method is invoked.

You can demarcate a block explicitly within the source code, or implicitly using the adapter configuration file. A block is demarcated explicitly when you set either a BATCH_START flag (sends the block using envelopes) or a TRANS_START flag (sends the block using transactions) is set in the current or preceding AdapterRow instances. The end of a block is demarcated by an AdapterRow instance with a BLOCK_END flag, or when an AdapterRow is sent to smart data streaming with the sendEnd() method instead of sendRow() or sendRowData().

If the AdapterRow instance is not explicitly part of a block, and the optional EspPublisher module property MaxPubPoolSize is set to a value greater than 1, the adapter framework automatically uses blocks to transmit the records. The MaxPubPoolSize parameter specifies the maximum size of the record pool before all records in the pool are published.

If the optional EspPublisher module property MaxPubPoolTime is configured, the record pool is published using blocks. MaxPubPoolTime specifies the maximum period of time, in milliseconds, for which records are pooled before being published.

If the threshold value of either MaxPubPoolSize or MaxPubPoolTime is reached, the record pool is published using blocks. A third optional configuration property, UseTransactions, controls whether the blocks are published using envelopes or transactions.

Records are published individually if event blocks are neither demarcated explicitly in the adapter code nor implicitly used based on the adapter configuration.

A typical transaction block using AdapterRow might look like this:

- An AdapterRow with a block flag that is set to TRANS_START
- Various AdapterRows with block flags that are set to BLOCK_DATA
- An AdapterRow with a block flag set to BLOCK_END

For AdapterRow instances with no data and only the start or end flag, set each data position in the instance to null. See the $STREAMING_HOME/adapters/framework/examples/src/com/sybase/esp/adapter/framework/examplemodules/ExampleRowInputTransporter.java example for publishing AdapterRow instances in various ways.

An AdapterRow instance cannot contain records from multiple transactions. If using blocks, records from an AdapterRow instance are published in one transaction or envelope.
2.7 Working with Schemas

Discover external schemas and create CCL schemas, streams, or windows based on the format of the data from the data source that is connected to an adapter.

Every row in a stream or window has the same structure (schema), including the column names and datatypes and the order in which the columns appear. Multiple streams or windows can use the same schema, but each stream or window can only have one schema.

Rather than manually creating a new schema in your smart data streaming project, you can use schema discovery to discover and automatically create a schema, stream, or window based on the format of the data from the data source to which your adapter connects. For example, if you create a table in your SAP HANA database, use the SAP HANA Output adapter to connect to the database. You can then use schema discovery to discover and create a schema, stream, or window in your smart data streaming project that corresponds to the schema of the table you created in your SAP HANA database.

While using discovery is a convenient way to create your CCL schema, pay attention to the datatypes that the CCL columns inherit from the external data source. Discovery tries to maintain the same level of precision, or greater, when mapping source datatypes to smart data streaming datatypes. Some databases, such as SAP IQ, support microsecond precision for the SQL_TIMESTAMP and SQL_TYPE_TIMESTAMP datatypes. As such, schema discovery maps these types to the smart data streaming datatype bigdatetime, which also supports microsecond precision. If your smart data streaming project does not require this level of precision, you can, after generating your schema through discovery, modify the schema to use a lower-precision datatype, such as msdate for millisecond precision.

To enable schema discovery, configure the properties of the adapters that support the feature.

In this section:

Implementing Schema Discovery in a Custom Adapter [page 105]
(Optional) Use interfaces and functions from the adapter toolkit to implement schema discovery in a transporter and formatter module. The two types of schema discovery are non-sampling and sampling. Use non-sampling schema discovery when the transporter can fully determine schema on its own. Use sampling schema discovery when the transporter cannot determine the schema and passes this data to the formatter to generate the schema.

Related Information

SAP HANA Smart Data Streaming: Configuration and Administration Guide
2.7.1 Implementing Schema Discovery in a Custom Adapter

(Optional) Use interfaces and functions from the adapter toolkit to implement schema discovery in a transporter and formatter module. The two types of schema discovery are non-sampling and sampling. Use non-sampling schema discovery when the transporter can fully determine schema on its own. Use sampling schema discovery when the transporter cannot determine the schema and passes this data to the formatter to generate the schema.

Procedure

1. Add the x_winCmdDisc (Windows) or x_unixCmdDisc (UNIX) parameter to the cnxml file for your custom adapter. See $STREAMING_HOME/adapters/framework/examples/discover/ExampleAdapterForDiscovery.cnxml for an example of a cnxml file with the discovery command.

2. Implement schema discovery in your custom modules:
   ○ (For transporter modules only) To implement non-sampling schema discovery, implement the com.sybase.esp.adapter.framework.discovery.TableDiscovery and com.sybase.esp.adapter.framework.discovery.ColumnDiscovery interfaces. For an example of an adapter with non-sampling schema discovery, see $STREAMING_HOME/adapters/framework/examples/discover. For the source code of a discoverable transporter module, see $STREAMING_HOME/adapters/framework/examples/src/com/Sybase/esp/adapter/framework/examplemodules/ExampleDiscoverableInputTransporter.java.
   ○ (For input adapters only) To implement sampling schema discovery:
     1. For the transporter module, implement the com.sybase.esp.adapter.framework.discovery.TableDiscoveryWithSample interface.
     2. For the formatter module, implement the com.sybase.esp.adapter.framework.discovery.ColumnDiscovery interface.

Task overview: Create a Custom Adapter [page 18]

Previous task: Enabling Guaranteed Delivery for an Input Transporter [page 106]

Next task: Configuring a New Adapter [page 118]

2.8 Guaranteed Delivery

Guaranteed delivery (GD) is a delivery mechanism used to guarantee that data is processed from a stream or window to an adapter.

GD ensures that data continues to be processed when:
• The smart data streaming server fails.
• The destination (third-party server) fails.
• The destination (third-party server) does not respond for a period of time.

Input adapters support GD using facilities provided by the external data source to which the input transporter connects.

In this section:

Enabling Guaranteed Delivery for an Input Transporter [page 106]

(Optional) Enable guaranteed delivery (GD) in a custom input transporter by implementing the com.sybase.esp.adapter.framework.event.AdapterRowEventListener interface, registering the GdAdapterEventListener class, and adding and setting the <GDMode> parameter to true for the EspPublisher or EspMultistreamPublisher.

2.8.1 Enabling Guaranteed Delivery for an Input Transporter

(Optional) Enable guaranteed delivery (GD) in a custom input transporter by implementing the com.sybase.esp.adapter.framework.event.AdapterRowEventListener interface, registering the GdAdapterEventListener class, and adding and setting the <GDMode> parameter to true for the EspPublisher or EspMultistreamPublisher.

Prerequisites

Create a custom input transporter module.

Procedure

1. In the adapter configuration file, add the <GDMode> parameter and set it to true for the EspPublisher or EspMultiStreamPublisher:

   ○ For EspPublisher:

   ```xml
   <EspPublisherParameters>
   <ProjectName>EspProject1</ProjectName>
   <StreamName>MyInStream</StreamName>
   <GDMode>true</GDMode>
   </EspPublisherParameters>
   ```

   ○ For EspMultiStreamPublisher:

   ```xml
   <EspMultiStreamPublisherParameters>
   <Streams>
   <Stream>
   <ProjectName>EspProject1</ProjectName>
   <StreamName>MyInStream1</StreamName>
   <GDMode>true</GDMode>
   </Stream>
   </Streams>
   </EspMultiStreamPublisherParameters>
   ```
2. Implement the com.sybase.esp.adapter.framework.event.AdapterRowEventListener interface. For example:

```java
public class GdAdapterEventListener implements AdapterRowEventListener {
    public void adapterEventPerformed(AdapterRowEvent event) {
        List<AdapterRow> rows = event.getAdapterRows();
        switch (event.getType()) {
            case PUBLISH_SUCCESS:
                processPublishSuccess();
                break;
            case PUBLISH_FAILURE:
                processPublishFailure();
                break;
            case FORMAT_FAILURE:
                processFormatFailure();
                break;
            default:
                break;
        }
    }
}
```

3. Create the class GdAdapterEventListener and register it to enable GD in the input transporter when it starts. For example:

```java
GdAdapterEventListener gdProcessor = new GdAdapterEventListener();
utility.getAdapterUtility().registerRowEventListener(gdProcessor,
    EventType.PUBLISH_SUCCESS);
utility.getAdapterUtility().registerRowEventListener(gdProcessor,
    EventType.PUBLISH_FAILURE);
utility.getAdapterUtility().registerRowEventListener(gdProcessor,
    EventType.FORMAT_FAILURE);
```

4. Keep track of the last row that published successfully to smart data streaming, either by using your external data source or the transporter itself.

   If you are using the input transporter, you can get the last row successfully published from the PUBLISH_SUCCESS message. If you are publishing in transactions or envelopes, the PUBLISH_SUCCESS message contains all rows in the last transaction or envelope to be successfully published. If you publish in single rows, the message contains the last single row that was successfully published.

**Task overview:** Create a Custom Adapter [page 18]

**Previous task:** Building a Custom Formatter Module [page 98]

**Next task:** Implementing Schema Discovery in a Custom Adapter [page 105]
2.9 EspConnector Modules

The EspConnector modules are responsible for connecting to smart data streaming. Connector module types include EspSubscriber, EspMultiStreamSubscriber, EspPublisher, and EspMultiStreamPublisher.

EspSubscriber subscribes to a stream in a project for smart data streaming and outputs data to the next module configured in the adapter configuration file (for example, a formatter or transporter). EspMultiStreamSubscriber has the same functionality but can subscribe to multiple streams.

EspPublisher takes data from a transporter module and publishes it to a stream in a project for smart data streaming. EspMultiStreamPublisher has the same functionality but can publish data to multiple streams.

In this section:

- **EspSubscriber Configuration** [page 109]
  The EspSubscriber module obtains data from the SAP HANA smart data streaming project and passes it along to a transporter or formatter module. Specify values for the EspSubscriber module in the adapter configuration file. Specify this module for an output adapter only.

- **EspMultiStreamSubscriber Configuration** [page 111]
  Specify values for the EspMultiStreamSubscriber module in the adapter configuration file. This module is specified only for an output adapter.

- **EspPublisher Configuration** [page 112]
  The EspPublisher module obtains data from a transporter or formatter module and publishes it to the SAP HANA smart data streaming project. Specify values for the EspPublisher module in the adapter configuration file. Specify this module for the input adapter only.

- **EspMultiStreamPublisher Configuration** [page 114]
  Specify values for the EspMultiStreamPublisher module in the adapter configuration file. This module is specified only for an input adapter.

Related Information

- Formatter Modules [page 59]
- Transporter Modules [page 20]
- Accessing Adapter Toolkit API Reference Information [page 19]
- Create a Custom Adapter [page 18]
- Debugging a Custom Adapter [page 140]
2.9.1 EspSubscriber Configuration

The EspSubscriber module obtains data from the SAP HANA smart data streaming project and passes it along to a transporter or formatter module. Specify values for the EspSubscriber module in the adapter configuration file. Specify this module for an output adapter only.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| ProjectName | Type: string  
(Required if running the adapter in unmanaged mode; optional if running in managed mode) The unique project tag of the smart data streaming project to which the adapter is connected. For example, StreamingProject2. This is the same project tag that you specify later in the adapter configuration file in the Name element within the smart data streaming (EspProjects) element. If you are starting the adapter with the smart data streaming project to which it is attached (running the adapter in managed mode), you can simply comment out this element as the adapter automatically connects to the project instance that started it. |
| StreamName  | Type: string  
(Required if running the adapter in unmanaged mode; optional if running in managed mode) The name of the stream from which the adapter subscribes to data. If you are starting the adapter with the smart data streaming project to which it is attached (running the adapter in managed mode), you can simply comment out this element as the adapter automatically connects to the project instance that started it. |
| OutputBase  | Type: boolean  
(Optional) If set to true, in addition to stream updates, the adapter outputs events that are in the stream or window when the adapter is started (the base content). If set to true and the adapter is running in GD mode, once the adapter has done a GD commit on the entire base content, the smart data streaming server does not redeliver the base content on adapter restart. Also, it only sends deltas that are saved for delivery. The default value is false. |
### OnlyBase

Type: boolean

(Advanced) Processes events that are in the stream or window when the adapter is started (the base content). Once it has processed the base content, the adapter stops and does not process any further message flow.

The default value is false. If set to true, outputBase is automatically set to true. Even if you set onlyBase to true and outputBase to false, the adapter automatically changes outputBase to true.

### EnableGDMode

Type: boolean

(Advanced) Indicates whether the adapter runs in guaranteed delivery (GD) mode. GD ensures that data continues to be processed in the case that the smart data streaming server fails, or the destination (third-party server) fails or does not respond for a long time. See Guaranteed Delivery in the SAP HANA Smart Data Streaming: Developer Guide for details on enabling GD for your project.

The default value is false.

### EnableGDCache

Type: boolean

(Advanced) If set to true, only those rows that can be recovered (that is, checkpointed) by the smart data streaming server on restart are sent to the end source. Other rows are cached internally by the adapter.

### GDSubscriberName

Type: string

(Advanced) If the adapter is running in GD mode (the EnableGDMode property is set to true), this element is a unique name to identify the GD subscription client. If this element is empty when running in GD mode, the adapter logs an error and shuts down.

### Related Information

- EspMultiStreamSubscriber Configuration [page 111]
- EspPublisher Configuration [page 112]
- EspMultiStreamPublisher Configuration [page 114]
- Smart Data Streaming Properties [page 116]
- Configuring a New Adapter [page 118]
- SAP HANA Smart Data Streaming: Developer Guide
2.9.2 EspMultiStreamSubscriber Configuration

Specify values for the EspMultiStreamSubscriber module in the adapter configuration file. This module is specified only for an output adapter.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Streams</td>
<td>(Required) Element containing the Stream, ProjectName, StreamName, and ColumnMapping elements.</td>
</tr>
<tr>
<td>Stream</td>
<td>(Required) Element containing details for the target project and streams to which the adapter is connected. Contains the ProjectName, StreamName, and ColumnMapping elements. You can specify multiple Stream elements.</td>
</tr>
</tbody>
</table>
| ProjectName | Type: string
(Required if running the adapter in unmanaged mode; optional if running in managed mode) The unique project tag of the smart data streaming project to which the adapter is connected. For example, StreamingProject2.
This is the same project tag that you specify later in the adapter configuration file in the Name element within the smart data streaming (EspProjects) element.
If you are starting the adapter with the smart data streaming project to which it is attached (running the adapter in managed mode), you can simply comment out this element as the adapter automatically connects to the project instance that started it. |
| StreamName  | Type: string
(Required if running the adapter in unmanaged mode; optional if running in managed mode) The name of the stream from which the adapter subscribes to data.
If you are starting the adapter with the smart data streaming project to which it is attached (running the adapter in managed mode), you can simply comment out this element as the adapter automatically connects to the project instance that started it. |
### Related Information

- [EspSubscriber Configuration](#)
- [EspPublisher Configuration](#)
- [EspMultiStreamPublisher Configuration](#)
- [Smart Data Streaming Properties](#)
- [Configuring a New Adapter](#)

### 2.9.3 EspPublisher Configuration

The EspPublisher module obtains data from a transporter or formatter module and publishes it to the SAP HANA smart data streaming project. Specify values for the EspPublisher module in the adapter configuration file. Specify this module for the input adapter only.

### XML Element Descriptions

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProjectName</td>
<td>Type: string</td>
</tr>
</tbody>
</table>
|                   | (Required if the adapter is running in unmanaged mode; optional if it is running in managed mode) The name of the smart data streaming project to which the adapter is connected. For example, StreamingProject2.
|                   | This is the same project tag that you specify later in the adapter configuration file in the Name element within the smart data streaming (EspProjects) element.
|                   | If you are starting the adapter with the smart data streaming project to which it is attached (that is, running the adapter in managed mode), you can simply comment out this element as the adapter automatically connects to the project instance that started it.

### Example

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColumnMapping</td>
<td>Type: string</td>
</tr>
</tbody>
</table>
|                   | (Optional) The column list of the source stream to which the adapter connects. Separate the columns with a space character. This element contains the "enumtype" attribute, which has these two valid values:
|                   | index The index of the column.
<p>|                   | name The column name.        |
|                   | The default value is name.   |</p>
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| StreamName       | Type: string  
(Required if the adapter is running in unmanaged mode; optional if it is running in managed mode) The name of the stream to which the adapter publishes data.  
If you are starting the adapter with the smart data streaming project to which it is attached (that is, running the adapter in managed mode), you can simply comment out this element as the adapter automatically connects to the project instance that started it. |
| MaxPubPoolSize   | Type: positive integer  
(Optional) The maximum size of the record pool. Record pooling, also referred to as block or batch publishing, allows for faster publication since there is less overall resource cost in publishing multiple records together, compared to publishing records individually.  
Record pooling is disabled if this value is 1. The default value is 256. |
| MaxPubPoolTime   | Type: positive integer  
(Optional) The maximum period of time, in milliseconds, for which records are pooled before being published. If not set, pooling time is unlimited and the pooling strategy is governed by maxPubPoolSize. No default value. |
| UseTransactions  | Type: boolean  
(Optional) If set to true, pooled messages are published to smart data streaming in transactions. If set to false, they are published in envelopes. The default value is false. |
| SafeOps          | Type: boolean  
(Advanced) Converts the opcodes INSERT and UPDATE to UPSERT, and converts DELETE to SAFEDELETE. The default value is false. |
| SkipDels         | Type: boolean  
(Advanced) Skips the rows with opcodes DELETE or SAFEDELETE. The default value is false. |
### Related Information

EspSubscriber Configuration [page 109]  
EspMultiStreamSubscriber Configuration [page 111]  
EspMultiStreamPublisher Configuration [page 114]  
Smart Data Streaming Properties [page 116]  
Configuring a New Adapter [page 118]  
Batch Processing [page 103]

### 2.9.4 EspMultiStreamPublisher Configuration

Specify values for the EspMultiStreamPublisher module in the adapter configuration file. This module is specified only for an input adapter.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_streams</td>
<td>(Required) Element containing the Stream, ProjectName, StreamName, and ColumnMapping elements.</td>
</tr>
</tbody>
</table>
| Stream              | (Required) Element containing details for the target project and streams to which the adapter is connected. Contains the ProjectName, StreamName, and ColumnMapping elements.  
You can specify multiple Stream elements. |
<p>| Filter              | (Optional) Element containing the MatchString element.                                                                                     |
| MatchString         | (Optional) Filters records with one or more column values. Contains a value attribute for specifying the value you want to filter by. No default value. |</p>
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| ProjectName         | **Type:** string  
(Required if the adapter is running in unmanaged mode; optional if it is running in managed mode) The name of the smart data streaming project to which the adapter is connected. For example, StreamingProject2.  
This is the same project tag that you specify later in the adapter configuration file in the Name element within the smart data streaming (EspProjects) element.  
If you are starting the adapter with the smart data streaming project to which it is attached (that is, running the adapter in managed mode), you can simply comment out this element as the adapter automatically connects to the project instance that started it. |
| StreamName          | **Type:** string  
(Required if the adapter is running in unmanaged mode; optional if it is running in managed mode) The name of the stream to which the adapter publishes data.  
If you are starting the adapter with the smart data streaming project to which it is attached (that is, running the adapter in managed mode), you can simply comment out this element as the adapter automatically connects to the project instance that started it. |
| MaxPubPoolSize      | **Type:** positive integer  
(Optional) The maximum size of the record pool. Record pooling, also referred to as block or batch publishing, allows for faster publication since there is less overall resource cost in publishing multiple records together, compared to publishing records individually.  
Record pooling is disabled if this value is 1. The default value is 256. |
| MaxPubPoolTime      | **Type:** positive integer  
(Optional) The maximum period of time, in milliseconds, for which records are pooled before being published. If not set, pooling time is unlimited and the pooling strategy is governed by maxPubPoolSize. No default value. |
| UseTransactions     | **Type:** boolean  
(Optional) If set to true, pooled messages are published to smart data streaming in transactions. If set to false, they are published in envelopes. The default value is false. |
### XML Element

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| ColumnMapping | Type: string  
(Optional) The column index list in the source row to be published to the target stream. Separate the columns using space character. If you do not set this element, all columns are published to the target stream. |
| SafeOps       | Type: boolean  
(Advanced) Converts the opcodes INSERT and UPDATE to UPSERT, and converts DELETE to SAFEDELETE. The default value is false. |
| SkipDels      | Type: boolean  
(Advanced) Skips the rows with opcodes DELETE or SAFEDELETE. The default value is false. |

### Related Information

EspSubscriber Configuration [page 109]  
EspMultiStreamSubscriber Configuration [page 111]  
EspPublisher Configuration [page 112]  
Smart Data Streaming Properties [page 116]  
Configuring a New Adapter [page 118]

### 2.10 Smart Data Streaming Properties

Edit the smart data streaming elements in the adapter configuration (.xml) file to connect a project to an adapter instance.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| EspProjects   | (Required only if the ProjectName element is specified in the EspConnector (publisher/subscriber) module and you are running the adapter in unmanaged mode)  
Element containing elements for connecting to smart data streaming. |
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EspProject</td>
<td>(Required only if the ProjectName element is specified in the EspConnector (publisher/subscriber) module and you are running the adapter in unmanaged mode) Element containing the Name and Uri elements. The information for the smart data streaming project to which the adapter is connected.</td>
</tr>
<tr>
<td>Name</td>
<td>Type: string (Required) The unique project tag of the smart data streaming project that the EspConnector (publisher/subscriber) module references.</td>
</tr>
<tr>
<td>Uri</td>
<td>Type: string (Required) The total project URI to connect to the smart data streaming project. For example, esps://&lt;host&gt;:3&lt;instance-number&gt;26/ws1/p1.</td>
</tr>
<tr>
<td>Security</td>
<td>(Required) Element containing all the authentication elements below. The details for the authentication method used for smart data streaming.</td>
</tr>
<tr>
<td>User</td>
<td>Type: string (Required) The username required to log in to smart data streaming (see AuthType). No default value.</td>
</tr>
<tr>
<td>Password</td>
<td>Type: string (Required) The password required to log in to smart data streaming (see AuthType). Includes an &quot;encrypted&quot; attribute indicating whether the Password value is encrypted. The default value is false. If &quot;encrypted&quot; is set to true and EncryptionAlgorithm is set to RSA, the password value is decrypted using RSAKeyStore and RSAKeyStorePassword. If &quot;encrypted&quot; is set to true and EncryptionAlgorithm is not set, the password is decrypted using the cipher key.</td>
</tr>
<tr>
<td>AuthType</td>
<td>Type: string (Required) Method used to authenticate to smart data streaming. Valid value is user_password for SAP HANA username and password authentication.</td>
</tr>
</tbody>
</table>
### Related Information

EspSubscriber Configuration [page 109]
EspMultiStreamSubscriber Configuration [page 111]
EspPublisher Configuration [page 112]
EspMultiStreamPublisher Configuration [page 114]
Configuring a New Adapter [page 118]

### 2.11 Configuring a New Adapter

Configure a new adapter by creating a configuration file for it. The configuration file defines the adapter component chain through which data is processed, and the connection to smart data streaming.

### Prerequisites

Create any custom transporters and formatters to use in this adapter instance.
Procedure

1. Create an <Adapter> element within which the following steps will add properties to.
2. Add a <Name> element and specify a name for the adapter instance.
3. Add a <Description> element and specify the purpose of the adapter.
4. (Optional) Add a <Log4jProperty> element and specify the full path to the log4j.properties logging file you wish to use.
5. Add a <Modules> element to contain all of the modules for your adapter instance.
6. For each module, specify:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceName</td>
<td>Type: string (Required) The instance name of the specific module to use. For example, &lt;MyInputTransporter&gt;.</td>
</tr>
<tr>
<td>Name</td>
<td>Type: string (Required) The name of the module as defined in the modulesdefine.xml or custommodulesdefine.xml file. This should be a unique name. For example, MyCustomInputTransporter.</td>
</tr>
<tr>
<td>Next</td>
<td>Type: string (Required if another module follows this one) Instance name of the module that follows this one.</td>
</tr>
<tr>
<td>BufferMaxSize</td>
<td>Type: integer (Advanced) The capacity of the buffer queue between this module and the next. The default value is 10240.</td>
</tr>
<tr>
<td>Parallel</td>
<td>Type: boolean (Optional; applies only to row-based formatters) If set to true, the module runs as a separated thread. If set to false, the module shares a thread with other modules. The default value is true.</td>
</tr>
<tr>
<td>Parameters</td>
<td>(Required) Parameters for the current module. For a custom module, the sub-element can reflect the name or type of the module, for example &lt;MyCustomInputTransporterParameters&gt;. EspPublisher, EspMultiStreamPublisher, EspSubscriber, and EspMultiStreamSubscriber all have set parameters that are configured specifically.</td>
</tr>
</tbody>
</table>

7. Configure one of these modules for your adapter.
For the EspPublisher, add a `<EspPublisherParameters>` sub-element and specify these parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ProjectName</strong></td>
<td>Type: <code>string</code>&lt;br&gt;(Required if the adapter is running in unmanaged mode; optional if it is running in managed mode)&lt;br&gt;The name of the smart data streaming project to which the adapter is connected. For example, StreamingProject2.&lt;br&gt;This is the same project tag that you specify later in the adapter configuration file in the <code>Name</code> element within the smart data streaming (<code>EspProjects</code>) element.&lt;br&gt;If you are starting the adapter with the smart data streaming project to which it is attached (that is, running the adapter in managed mode), you can simply comment out this element as the adapter automatically connects to the project instance that started it.</td>
</tr>
<tr>
<td><strong>EspPublisherParameters</strong></td>
<td>(Required) The element containing elements for the ESP publisher.</td>
</tr>
<tr>
<td><strong>StreamName</strong></td>
<td>Type: <code>string</code>&lt;br&gt;(Required if the adapter is running in unmanaged mode; optional if it is running in managed mode)&lt;br&gt;The name of the stream to which the adapter publishes data.&lt;br&gt;If you are starting the adapter with the smart data streaming project to which it is attached (that is, running the adapter in managed mode), you can simply comment out this element as the adapter automatically connects to the project instance that started it.</td>
</tr>
<tr>
<td><strong>MaxPubPoolSize</strong></td>
<td>Type: <code>positive integer</code>&lt;br&gt;(Optional) The maximum size of the record pool. Record pooling, also referred to as block or batch publishing, allows for faster publication since there is less overall resource cost in publishing multiple records together, compared to publishing records individually.&lt;br&gt;Record pooling is disabled if this value is 1. The default value is 256.</td>
</tr>
<tr>
<td><strong>MaxPubPoolTime</strong></td>
<td>Type: <code>positive integer</code>&lt;br&gt;(Optional) The maximum period of time, in milliseconds, for which records are pooled before being published. If not set, pooling time is unlimited and the pooling strategy is governed by <code>maxPubPoolSize</code>. No default value.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| UseTransactions  | Type: boolean  
(Optional) If set to true, pooled messages are published to smart data streaming in transactions. If set to false, they are published in envelopes. The default value is false. |
| SafeOps          | Type: boolean  
(Advanced) Converts the opcodes INSERT and UPDATE to UPSERT, and converts DELETE to SAFEDELETE. The default value is false.            |
| SkipDels         | Type: boolean  
(Advanced) Skips the rows with opcodes DELETE or SAFEDELETE. The default value is false.                                              |

For EspMultiStreamPublisher, add a `<EspMultiStreamPublisherParameters>` sub-element and specify:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Streams</td>
<td>(Required) Section containing the Stream, ProjectName, StreamName, and ColumnMapping parameters.</td>
</tr>
</tbody>
</table>
| Stream           | (Required) Section containing details for the target project and streams to which the adapter is connected.  
Contains the ProjectName, StreamName, and ColumnMapping parameters.  
You can specify multiple Stream sections. |
| Filter           | (Optional) Section containing the MatchString parameter.                                                                                   |
| MatchString      | (Optional) Filters records with one or more column values. Contains a value attribute for specifying the value you want to filter by. No default value. |
| ProjectName      | Type: string  
(Required) The name of the smart data streaming project to which the adapter belongs. This is the same project as specified in the Name parameter of the smart data streaming projects section of the adapter configuration file. |
| StreamName       | Type: string  
(Required) The name of the target stream to which the adapter connects.                                                                     |
Parameter | Description
---|---
ColumnMapping | Type: string
(Optional) The column index list in the source row to be published to the target stream. Separate the columns using space character. If you do not set this parameter, all columns are published to the target stream.

For EspSubscriber, add a `<EspSubscriberParameters>` sub-element and specify these parameters:

Parameter | Description
---|---
ProjectName | Type: string
(Required if running the adapter in unmanaged mode; optional if running in managed mode) The unique project tag of the smart data streaming project to which the adapter is connected. For example, StreamingProject2.
This is the same project tag that you specify later in the adapter configuration file in the `Name` element within the smart data streaming (`EspProjects`) element.
If you are starting the adapter with the smart data streaming project to which it is attached (running the adapter in managed mode), you can simply comment out this element as the adapter automatically connects to the project instance that started it.

StreamName | Type: string
(Required if running the adapter in unmanaged mode; optional if running in managed mode) The name of the stream from which the adapter subscribes to data.
If you are starting the adapter with the smart data streaming project to which it is attached (running the adapter in managed mode), you can simply comment out this element as the adapter automatically connects to the project instance that started it.

For EspMultiStreamSubscriber, add a `<EspMultiStreamSubscriberParameters>` sub-element and specify these parameters:

Parameter | Description
---|---
Streams | (Required) Section containing the `Stream`, `ProjectName`, `StreamName`, and `ColumnMapping` parameters.
### Parameter | Description
---|---
Stream | (Required) Section containing details for the target project and streams to which the adapter is connected. Contains the ProjectName, StreamName, and ColumnMapping parameters. You can specify multiple Stream elements.

### ProjectName | Type: string
| (Required) The name of the smart data streaming project to which the adapter belongs. This is the same project as specified in the Name parameter of the smart data streaming projects section of the adapter configuration file.

### StreamName | Type: string
| (Required) The name of the source stream to which the adapter connects.

### ColumnMapping | Type: string
| (Optional) The column list of the source stream to which the adapter connects. Separate the columns with a space character. This parameter contains the "enumtype" attribute, which has these valid values:

- `index` The index of the column.
- `name` The column name.

The default value is name.

8. Specify a connection to smart data streaming:

### XML Element | Description
---|---
EspProjects | (Required only if the ProjectName element is specified in the EspConnector (publisher/subscriber) module and you are running the adapter in unmanaged mode)
Element containing elements for connecting to smart data streaming.

EspProject | (Required only if the ProjectName element is specified in the EspConnector (publisher/subscriber) module and you are running the adapter in unmanaged mode)
Element containing the Name and Uri elements. The information for the smart data streaming project to which the adapter is connected.
<table>
<thead>
<tr>
<th>XML Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Name**    | Type: `string`  
(Required) The unique project tag of the smart data streaming project that the EspConnector (publisher/ subscriber) module references. |
| **Uri**     | Type: `string`  
(Required) The total project URI to connect to the smart data streaming project. For example, esps:// <host>:3<instance-number>26/ws1/p1. |
| **Security**| (Required) Element containing all the authentication elements below. The details for the authentication method used for smart data streaming. |
| **User**    | Type: `string`  
(Required) The username required to log in to smart data streaming (see AuthType). No default value. |
| **Password**| Type: `string`  
(Required) The password required to log in to smart data streaming (see AuthType). Includes an "encrypted" attribute indicating whether the Password value is encrypted. The default value is false. If "encrypted" is set to true and EncryptionAlgorithm is set to RSA, the password value is decrypted using RSAKeyStore and RSAKeyStorePassword. If "encrypted" is set to true and EncryptionAlgorithm is not set, the password is decrypted using the cipher key. |
| **AuthType**| Type: `string`  
(Required) Method used to authenticate to smart data streaming. Valid value is `user_password` for SAP HANA username and password authentication. |
| **RSAKeyStore** | Type: `string`  
(Dependent required) The location of the RSA keystore, and decrypts the password value. |
9. (Optional) Add a GlobalParameters element. This node is visible to all modules that are configured within the adapter configuration file:

a. Define XML schema for the GlobalParameter in the $STREAMING_CUSTOM_ADAPTERS_HOME/config/parametersdefine.xsd file.

b. Call the Utility.getGlobalParameterNode() or utility.getGlobalParameters() function to get the XML object for this node.

Example

Here is an example of a configuration file for the Socket JSON Input adapter:

```xml
<?xml version="1.0" encoding="utf-8"?>
<Adapter>
  <Name>socket_json_input</Name>
  <Description>An adapter that receives JSON message from socket server, transforms to smart data streaming data format, and publishes to the stream.</Description>
  <Log4jProperty>./log4j.properties</Log4jProperty>
  <Modules>
    <Module type="transporter">
      <InstanceName>MyExampleSocketInTransporter</InstanceName>
      <Name>SocketInputTransporter</Name>
      <Next>MyJsonStreamToJsonStringFormatter</Next>
      <Parameters>
        <SocketInputTransporterParameters>
          <Host>host</Host>
          <Port>9998</Port>
          <EpFile/>
          <Retryperiod>60</Retryperiod>
          <Blocksize>512</Blocksize>
          <KeepAlive>true</KeepAlive>
        </SocketInputTransporterParameters>
      </Parameters>
    </Module>
    <Module type="formatter">
      <InstanceName>MyJsonStreamToJsonStringFormatter</InstanceName>
      <Name>JsonStreamToJsonStringFormatter</Name>
      <Next>MyJsonInFormatter</Next>
      <Parameters/>
    </Module>
    <Module type="formatter">
      <InstanceName>MyJsonInFormatter</InstanceName>
      <Name>JsonInFormatter</Name>
      <Next>
        <Parameters/>
      </Next>
    </Module>
  </Modules>
</Adapter>
```
<Name>JsonStringToEspFormatter</Name>
<Next>MyInStream_Publisher</Next>
<Parameters>
  <JsonStringToEspFormatterParameters>
    <SecondDateFormat>yyyy-MM-dd HH:mm:ss.SSS</SecondDateFormat>
    <MSDateFormat>yyyy/MM/dd HH:mm:ss</MSDateFormat>
    <esps://localhost:19011>/Port
    <PortColumnMappings>>
      9998</Port><ColsMapping streamname="EntityStream"
      rootpath="entities">
      <Column
        <EpFile><>display_text</Column>
        <Column>domain_role</Column>/EpFile>
      <Retryperiod>60</Retryperiod>
      <Blocksize>Column512)</Blocksize>
      <KeepAlive>offset=true</KeepAlive>
      </ColsMapping>
    </JsonStringToEspFormatterParameters>
  </Parameters>
</Module>
<Module type="formatter">
  <InstanceName>MyJsonStreamToJsonStringFormatter</InstanceName>
  <Name>JsonStreamToJsonStringFormatter</Column>
  <Columns>
    <Column>length</Column>
  </Columns>
  <ColsMapping>
    <JsonStringToEspFormatterParameters>
      <Parameters>
      </Parameters>
    </JsonStringToEspFormatterParameters>
  </ColsMapping>
</Module>
<Module type="espconnector">
  <InstanceName>MyInStream_Publisher</InstanceName>
  <Name>EspPublisher</Name>
  <Parameters>
    <EspPublisherParameters>
      <ProjectName>EspProject1</ProjectName>
      <StreamName>EntityStream</StreamName>
      <MaxPubPoolSize/>1</MaxPubPoolSize>
      <UseTransactions>false</UseTransactions>
      <SafeOps>true</SafeOps>
      <SkipDels>true</SkipDels>
      </EspPublisherParameters>
  </Parameters>
</Module>
</Modules>
</EspProjects>
</EspProject>
<Name>EspProject1</Name>
<Uri>esps://<host>:3<instance-number>26null/sample_workspace/socket_json_input</Uri>
<Security>
  <User></User>
  <Password encrypted="false"></Password>
  <AuthType>user_password</AuthType>
  <RSAKeyStorePassword>Sybase123</RSAKeyStorePassword>
</Security>
</EspProject>
</EspProjects>
</GlobalParameters>
</Adapter>

In this section:
Encrypting Adapter Configuration Files [page 127]
Use the streamingencrypt executable to encrypt parameters within adapter configuration files.

Adapter Logging Configuration [page 129]
Adapters that are currently available from SAP use the log4j API to log errors, warnings, and debugging messages. A sample log4j.properties file containing the logging configuration is located in the \%STREAMING_HOME%\adapters\framework\config or \%STREAMING_HOME%\adapters\<adapter-name>\config directory.

Task overview: Create a Custom Adapter [page 18]

Previous task: Implementing Schema Discovery in a Custom Adapter [page 105]

Next task: Creating a cnxml File for a Custom Adapter [page 158]

Related Information

EspSubscriber Configuration [page 109]
EspMultiStreamSubscriber Configuration [page 111]
EspPublisher Configuration [page 112]
EspMultiStreamPublisher Configuration [page 114]
Smart Data Streaming Properties [page 116]

2.11.1 Encrypting Adapter Configuration Files

Use the streamingencrypt executable to encrypt parameters within adapter configuration files.

Context

Some parameters use an older encryption mechanism. If you have any of the following encrypted already, it is not necessary to re-encrypt these parameters:

- Password
- RSAKeystore
- RSAKeystorePassword
- RSAResidKeyStoreAlias

All other parameters use the streamingencrypt utility.
Procedure

1. Use a text editor to open the required adapter configuration file:
   `STREAMING_HOME/adapters/framework/instances/<adapter-name>/adapter_config.xml`

   **Note**
   The SAP RFC Input and Output adapters, and the Web Services (SOAP) Input and Output adapters, do not have default configuration files. Use the provided examples to create your own adapter_config.xml for these adapters:
   `STREAMING_HOME/adapters/<adapter-name>/examples/<example-name>/adapter_config.xml`

2. Add the `encrypted` attribute to the parameter you want to encrypt and set it to `encrypted="true"`. This attribute ensures that the server recognizes the value as encrypted text and decrypts it at runtime. If the attribute is not set to true, the server does not recognize the value as encrypted text and tries to process the value without decrypting it, resulting in errors.

   In the following section of a sample adapter configuration file, encrypt the value "User123":

   ```xml
   <Adapter>
   <Cipher>
     <File>$STREAMING_CUSTOM_ADAPTERS_HOME/adapter.key</File>
   </Cipher>
   .
   .
   <EspProjects>
     <ESP Project>
     <Security>
       <User encrypted="true">User123</User>
       <Password encrypted="true">myPassword123</Password>
       <AuthType>user_password</AuthType>
     </Security>
     </ESP Project>
   </EspProjects>
   </Adapter>
   ```

3. Update the parameter definition in the relevant `.xsd` file in the `/config` directory. In this case, use a text editor to open `STREAMING_HOME/adapters/framework/config/framework.xsd`.

   Original definition:
   ```xml
   <xs:element name="User" type="xs:string" minOccurs="0" maxOccurs="1"/>
   ```

   Required definition:
   ```xml
   <xs:element name="User" minOccurs="0" maxOccurs="1">
     <xs:complexType>
       <xs:simpleContent>
         <xs:extension base="xs:string">
           <xs:attribute name="encrypted" type="xs:boolean" default="false"/>
         </xs:extension>
       </xs:simpleContent>
     </xs:complexType>
   </xs:element>
   ```
4. **Note** the value in the Cipher element in step 2 [page 128]. This is the key file required to encrypt values in adapter configuration files. Optionally, create a new adapter key:
   a. From a command line, navigate to `STREAMING_HOME/bin` and launch the `streamingencrypt` executable using the `--create-key` option:

   ```
   streamingencrypt --create-key adapter.key
   ```

   The command writes a new key to the file `adapter.key`.
   b. Add the Cipher element to `adapter_config.xml` file using the format in step 2 [page 128].

5. From a command line, navigate to `STREAMING_HOME/bin` and launch the `streamingencrypt` executable using the `--encrypt` option:

   ```
   streamingencrypt --encrypt <key-file> --text <text>
   ```

   If you enter the `--text` value successfully, the `streamingencrypt` executable writes the encrypted text to the display.

6. Copy and paste the encrypted text from the utility into the adapter configuration file you opened in step 1 [page 128]. Replace the original value in the parameter with the encrypted text.

7. Save and close the adapter configuration file.

2.11.2 **Adapter Logging Configuration**

Adapters that are currently available from SAP use the `log4j` API to log errors, warnings, and debugging messages. A sample `log4j.properties` file containing the logging configuration is located in the `%STREAMING_HOME%/adapters/framework/config` or `%STREAMING_HOME%/adapters/<adapter-name>/config` directory.

Specify the location of the logging file you wish to use in the `Log4jProperty` parameter within an adapter configuration file. For example, in a configuration file for the File/Hadoop JSON Output adapter:

```xml
<Adapter>
  <Name>File/Hadoop JSON Output</Name>
  <Description>Transforms SDS data to JSON output</Description>
  <Log4jProperty>./log4j.properties</Log4jProperty>
</Adapter>
```

You can modify the logging levels within that file or the `log4j.properties` file, which is used by default.

Set the `ADAPTER_CLASSPATH` environment variable to point to the configuration directory of each adapter for which you are configuring logging.

The logging levels in `log4j.properties` are:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>Logs no events.</td>
</tr>
<tr>
<td>Level</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FATAL</td>
<td>Logs severe errors that prevent the application from continuing.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Logs potentially recoverable application errors.</td>
</tr>
<tr>
<td>WARN</td>
<td>Logs events that possibly lead to errors.</td>
</tr>
<tr>
<td>INFO</td>
<td>Logs events for informational purposes.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>Logs general debugging events.</td>
</tr>
<tr>
<td>TRACE</td>
<td>Logs fine-grained debug messages that capture the flow of the application.</td>
</tr>
<tr>
<td>ALL</td>
<td>Logs all events.</td>
</tr>
</tbody>
</table>

**Note**

Setting the log level to `DEBUG` or `ALL` may result in large log files. The default value is `INFO`.  

Here is a sample `log4j.properties` file:

```
# Set root logger level to INFO and set appenders to stdout, file and email
log4j.rootLogger=INFO, stdout, R
# stdout appender
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{MM-dd-yyyy HH:mm:ss.SSS} %p [%t] (%C{l}.%M) %m%n
log4j.appender.stdout.Threshold=INFO
# file appender
log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
log4j.appender.r.File=logs/adapter.log
log4j.appender.R.DatePattern='.'yyyy-MM-dd
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%d{MM-dd-yyyy HH:mm:ss.SSS} %p [%t] (%C{l}.%M) %m%n
log4j.appender.R.Threshold=INFO
# email appender
log4j.appender.email=org.apache.log4j.net.SMTPAppender
log4j.appender.email.To=your.name@yourcompany.com
log4j.appender.email.From=alert.manager@yourcompany.com
log4j.appender.email.SMTPHost=yourmailhost
log4j.appender.email.BufferSize=1
log4j.appender.email.Subject=Adapter Error
log4j.appender.email.layout=org.apache.log4j.PatternLayout
log4j.appender.email.layout.ConversionPattern=%d{MM-dd-yyyy HH:mm:ss.SSS} %p [%t] (%C{l}.%M) %m%n
log4j.appender.email.Threshold=ERROR
log4j.logger.com.sybase.esp=INFO
```

The `log4j.rootLogger` option sets the default log behavior for all the sub-loggers in the adapter. In addition to the root logger, the adapter contains various sub-loggers that control logging for specific adapter functions.

Setting the `log4j.rootLogger` to any value more verbose than `INFO` may produce excess information. If you explicitly set the log level for a sub-logger, you overwrite the default setting for that particular logger. In this way, you can make sub-loggers more verbose than the default. The names for smart data streaming-related loggers contain the string `com.sybase.esp`.  

---

SAP HANA Smart Data Streaming: Building Custom Adapters

Smart Data Streaming Adapter Toolkit
2.12 Starting an Adapter

Start the adapter either in unmanaged, managed, or cluster-managed mode. In unmanaged and cluster-managed mode, the adapter is started separately from the smart data streaming project, and in managed mode, the adapter is started with the smart data streaming project.

Prerequisites

- Edit the `<ESPProjects>` section in the adapter (.xml) configuration file to specify details for the cluster.
- Start the node. Log on to any host as the `<sid>`adm user and run:

```
sapcontrol -nr <instance#> -function InstanceStart <streaming-hostname> <instance#>
```

Procedure

<table>
<thead>
<tr>
<th>Mode</th>
<th>Steps</th>
</tr>
</thead>
</table>
| Unmanaged, from the command line | 1. Open a terminal window and navigate to `STREAMING_HOME/adapters/framework/instances/<adapter-name>`.  
   i Note  
   STREAMING_HOME refers to the environment variable. In Windows, enter `%STREAMING_HOME%`. In Unix, enter `$STREAMING_HOME`.  
   2. Enter `start_adapter.bat <adapter-config-file>` (Windows) or `./start_adapter.sh <adapter-config-file>` (Unix). |
| Unmanaged, using the start command files | 1. Start the node. Log on to any host as the `<sid>`adm user and run:  
   ```
sapcontrol -nr <instance#> -function InstanceStart <streaming-hostname> <instance#>
```
   2. Start the adapter by running `start_adapter.bat` (Windows) or `start_adapter.sh` (Unix). |
| Managed | 1. Add an ATTACH ADAPTER statement in the CCL file of your project to reference the adapter. For example:  
   ```
   ATTACH INPUT ADAPTER Generic_Input_Adapter__external_1  
   TYPE genericinputadapter  
   to BaseInput  
   PROPERTIES configFilePath = 'adapter_config.xml';
   ``` |
### Cluster-Managed Mode

1. **Start the node.** Log on to any host as the `<sid>adm` user and run:

   ```
sapcontrol -nr `<instance#>` -function InstanceStart `<streaming-hostname>` `<instance#>`
   ```

2. **Add the adapter to the cluster:**

   ```
STREAMING_HOME/bin/streamingclusteradmin --uri=esps://<host>:3<instance-number>26
   --username=<username> --password=<password> --
   add_adapter --workspace-name= `<workspace-name>` --project-name=`project-name` --
   --arc=`adapter-runtime-config` --adc=`application-deployment-config`
   ```

3. **Deploy the project on the cluster:**

   ```
STREAMING_HOME/bin/streamingclusteradmin --uri=esps://<host>:3<instance-number>26
   --username=<username> --password=<password> --
   add_project --workspace-name= `<workspace-name>` --project-name=`project-name` --
   ccx=`model-name`.ccx
   ```

4. **Start the project:**

   ```
streamingclusteradmin --uri=esps://<host>:3<instance-number>26
   --username=<username> --password=<password> --
   start_project --workspace-name=`workspace-name` --project-name=`project-name`
   ```

---

### Steps

2. **Start the node.** Log on to any host as the `<sid>adm` user and run:

   ```
sapcontrol -nr `<instance#>` -function InstanceStart `<streaming-hostname>` `<instance#>`
   ```

3. **Compile CCL to create CCX:**

   ```
STREAMING_HOME/bin/streamingcompiler -i `<model-name>.ccl` -o `<model-name>.ccx`
   ```

4. **Deploy the project on the cluster:**

   ```
STREAMING_HOME/bin/streamingclusteradmin --uri=esps://<host>:3<instance-number>26
   --username=<username> --password=<password> --
   add_project --workspace-name=`workspace-name` --project-name=`project-name` --
   ccx=`model-name`.ccx
   ```

5. **Start the deployed project on the cluster:**

   ```
STREAMING_HOME/bin/streamingclusteradmin --uri=esps://<host>:3<instance-number>26
   --username=<username> --password=<password> --
   start_project --
   workspace-name=`workspace-name` --project-name=`project-name`
   ```
### Task overview: Create a Custom Adapter [page 18]

### Previous task: Creating a cnxml File for a Custom Adapter [page 158]

### Next task: Stopping an Adapter [page 133]

### Related Information

- Stopping an Adapter [page 133]
- Adapter Integration Framework [page 157]
- Adapter Toolkit: Sample cnxml File for Output Adapters [page 139]
- Adapter Toolkit: Sample cnxml File for Input Adapters [page 138]

## 2.13 Stopping an Adapter

Stop the adapter either in unmanaged, managed, or cluster-managed mode. In unmanaged and cluster-managed mode, the adapter is stopped separately from the smart data streaming project. In managed mode, the adapter is stopped with the smart data streaming project.

### Procedure

<table>
<thead>
<tr>
<th>Mode</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unmanaged, from the command line</td>
<td>1. Open a terminal window and navigate to <code>STREAMING_HOME/adapters/framework/instances/&lt;adapter-name&gt;</code>.</td>
</tr>
</tbody>
</table>
### Mode

<table>
<thead>
<tr>
<th>Mode</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Note</strong></td>
<td>STREAMING_HOME refers to the environment variable. In Windows, enter</td>
</tr>
<tr>
<td></td>
<td>%STREAMING_HOME%. In Unix, enter $STREAMING_HOME.</td>
</tr>
<tr>
<td></td>
<td>2. Enter <code>stop_adapter.bat &lt;adapter-config-file&gt;</code> (Windows) or ./</td>
</tr>
<tr>
<td></td>
<td><code>stop_adapter.sh &lt;adapter-config-file&gt;</code> (Unix)</td>
</tr>
</tbody>
</table>

#### Unmanaged

1. Stop the node. Log on to any host as the `<sid>`adm user and run:

   ```
   sapcontrol -nr <instance#> -function InstanceStop
   <streaming-hostname> <instance#>
   ```

2. Stop the adapter by running `stop_adapter.bat` or `stop_adapter.sh`.

#### Managed

1. Stop the project to which the adapter is attached:

   ```
   STREAMING_HOME/bin/streamingclusteradmin
   --uri=esps://<host>:3<instance-number>26
   --username=<username> --password=<password> --
   stop project <workspace-name>/<project-name>
   ```

   You can also stop the adapter by issuing the `stop adapter` command from the `streamingprojectclient` tool.

#### Cluster-Managed Mode

1. Stop the adapter:

   ```
   STREAMING_HOME/bin/streamingclusteradmin --uri=esps://
   <host>:3<instance-number>26
   --username=<username> --password=<password> --
   stop_adapter --
   workspace-name=<workspace-name> --adapter-
   name=<adapter-name>
   ```

---

**Task overview:** Create a Custom Adapter [page 18]

**Previous task:** Starting an Adapter [page 131]

**Related Information**

- Starting an Adapter [page 131]
- Adapter Integration Framework [page 157]
- Adapter Toolkit: Sample cnxml File for Output Adapters [page 139]
- Adapter Toolkit: Sample cnxml File for Input Adapters [page 138]
2.14 Adapter Toolkit Examples

The `%STREAMING_HOME%/adapters/framework/examples` directory contains various example modules for the adapter toolkit.

<table>
<thead>
<tr>
<th>Folder Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discover</td>
<td>Contains an example adapter that has schema discovery implemented.</td>
</tr>
<tr>
<td></td>
<td>Before running this example, copy the ExampleAdapterForDiscovery.cnxml file to the <code>$STREAMING_HOME/lib/adapters</code> directory.</td>
</tr>
<tr>
<td>dual_direction_transporter</td>
<td>Contains an example adapter that concurrently processes input and output data.</td>
</tr>
<tr>
<td>esp_pipe</td>
<td>Contains an example adapter that pipes two smart data streaming projects together.</td>
</tr>
<tr>
<td>multistream_publisher</td>
<td>Contains an example adapter that uses the multistream publisher module.</td>
</tr>
<tr>
<td>multistream_subscriber</td>
<td>Contains an example adapter that uses the multistream subscriber module.</td>
</tr>
<tr>
<td>output</td>
<td>Contains an example output adapter that sends smart data streaming data to the console.</td>
</tr>
<tr>
<td>polling_input</td>
<td>Contains an example of a polling input adapter.</td>
</tr>
<tr>
<td>row_input</td>
<td>Contains an example input adapter that publishes pooled messages in transactions.</td>
</tr>
<tr>
<td>single_inputtransporter</td>
<td>Contains an example adapter that has only two modules: a transporter and an Espconnector.</td>
</tr>
<tr>
<td>src</td>
<td>Contains the source code for the example modules.</td>
</tr>
<tr>
<td>streaming_input</td>
<td>Contains an example streaming input adapter.</td>
</tr>
<tr>
<td>streaming_output</td>
<td>Contains an example streaming output adapter.</td>
</tr>
<tr>
<td>stringlist_input</td>
<td>Contains an example adapter that uses the String List to ESP Formatter.</td>
</tr>
<tr>
<td>stringlist_output</td>
<td>Contains an example adapter that uses the ESP to String List Formatter.</td>
</tr>
</tbody>
</table>

In this section:

Running an Adapter Example [page 136]
Use the sample content within the $STREAMING_HOME/adapters/framework/examples directory to run an adapter example.

**Running the Schema Discovery Adapter Example [page 137]**
Use the sample content within the $STREAMING_HOME/adapters/framework/examples/discover directory to run the schema discovery adapter example.

## 2.14.1 Running an Adapter Example

Use the sample content within the $STREAMING_HOME/adapters/framework/examples directory to run an adapter example.

### Prerequisites

1. Stop any other external adapter that is using one of the transporter or formatter modules provided by SAP.

2. **Back up the**
   - $STREAMING_CUSTOM_ADAPTERS_HOME\config\custommodulesdefine.xml (Windows) or $STREAMING_CUSTOM_ADAPTERS_HOME/config/custommodulesdefine.xml (Unix) file.

3. **Copy**
   - $STREAMING_HOME\adapters\framework\examples\custommodulesdefine.xml (Windows) or $STREAMING_HOME/adapters/framework/examples/custommodulesdefine.xml (Unix) to the $STREAMING_CUSTOM_ADAPTERS_HOME\config (Windows) or $STREAMING_CUSTOM_ADAPTERS_HOME/config (Unix) directory.

4. **Edit the**
   - set_example_env.bat (Windows) or set_example_env.sh (Unix) script and the adapter_config.xml file for each example to specify correct values for the user name and password parameters.

### Context

These steps are applicable to all examples except from the schema discovery example located in the

- $STREAMING_HOME\adapters\framework\examples\discover (Windows) or $STREAMING_HOME/adapters/framework/examples/discover (Unix) directory.

### Procedure

1. **Start the node.** Log on to any host as the `<sid>adm` user and run:
   ```bash
   sapcontrol -nr instance# -function InstanceStart streaming-hostname instance#
   ```

2. **Start the smart data streaming project by executing**
   ```bash
   start_project.bat (Windows) or start_project.sh (Unix).
   ```
3. Start a subscription by using one of the various `subscribe*.bat` (Windows)/`subscribe*.sh` (Unix) or `statistic_subscrib*.bat` (Windows)/`statistic_subscrib*.sh` (Unix) scripts to validate the results of the adapter.

4. Start the adapter by running the `start_adapter.bat` (Windows) or `start_adapter.sh` (Unix) commands.

5. Run one of the examples in the `%STREAMING_HOME%\adapters\framework\examples` (Windows) or `$STREAMING_HOME/adapters/framework/examples` (Unix) directory.

Next Steps

Restore the `%STREAMING_CUSTOM_ADAPTERS_HOME%\config\custommodulesdefine.xml` (Windows) or `$STREAMING_CUSTOM_ADAPTERS_HOME/config/custommodulesdefine.xml` (Unix) file.

2.14.2 Running the Schema Discovery Adapter Example

Use the sample content within the `$STREAMING_HOME/adapters/framework/examples/discover` directory to run the schema discovery adapter example.

Prerequisites

1. Stop any other external adapter that is using one of the transporter or formatter modules provided by SAP.

2. Back up the `%STREAMING_CUSTOM_ADAPTERS_HOME%\config\custommodulesdefine.xml` (Windows) or `$STREAMING_CUSTOM_ADAPTERS_HOME/config/custommodulesdefine.xml` (Unix) file.

3. Copy `%STREAMING_HOME%\adapters\framework\examples\custommodulesdefine.xml` (Windows) or `$STREAMING_HOME/adapters/framework/examples/custommodulesdefine.xml` (Unix) to the `%STREAMING_CUSTOM_ADAPTERS_HOME%\config` (Windows) or `$STREAMING_CUSTOM_ADAPTERS_HOME/config` (Unix) directory.

4. Edit the `set_example_env.bat` (Windows) or `set_example_env.sh` (Unix) script and the `adapter_config.xml` file for each example to specify correct values for the user name and password parameters.

Procedure

1. Start the SAP HANA studio.

2. In the SAP HANA Streaming Development perspective, drag and drop the Example Adapter for schema discovery onto the canvas.
3. Edit the adapter’s Adapter Directory Path property and point it to the directory where the discovery example resides. For example, %STREAMING_HOME%/adapters/framework/examples/discover (Windows) or $STREAMING_HOME/adapters/framework/examples/discover (Unix).

4. Click the adapter’s schema discovery icon and complete the steps in the discovery wizard to add a new element to the project matching the discovered schema.

Next Steps

Restore the %STREAMING_CUSTOM_ADAPTERS_HOME%/config/custommodulesdefine.xml (Windows) or $STREAMING_CUSTOM_ADAPTERS_HOME/config/custommodulesdefine.xml (Unix) file.

2.15 Adapter Toolkit: Sample cnxml File for Input Adapters

**Adapter type:** toolkit_input. Sample cnxml file for an input adapter created using the adapter toolkit. You can use this file as reference for creating your own cnxml file for your custom adapter.

This file is located in the $STREAMING_HOME/lib/adapters directory. The cnxml files are required if you want to manage your custom external adapter using the smart data streaming server. Set these properties in the studio adapter properties dialog.

<table>
<thead>
<tr>
<th>Property Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapter Configuration File</td>
<td>Property ID: configFilePath</td>
</tr>
<tr>
<td></td>
<td>Type: filename</td>
</tr>
<tr>
<td></td>
<td>(Required) Specify the path to the adapter configuration file.</td>
</tr>
</tbody>
</table>

If you use the ATTACH ADAPTER statement to attach an adapter, supply the adapter type. If you use the ATTACH ADAPTER statement with the PROPERTIES clause to override parameter values in the adapter configuration file, add the following to the adapter cnxml file:

Include the PropertySet parameter.

```xml
<Parameter id="propertyset"
  label="propertyset"
  descr="to look up properties in project configuration"
  type="string"
  use="advanced"
  default="" &quot;/>
```

Pass the parameter-value pair as Java System Property format.

Unix:

```xml
<Internal id="x_unixCmdExec"
  label="Execute Command"
  type="string"
  "/>
Windows:

```xml
<Internal id="x_winCmdExec"
  label="Execute Command"
  type="string"
  default="&quot;$STREAMING_HOME/adapters/framework/bin/start.bat&quot; &quot;/&quot;adapter_config.xml&quot; &quot;&quot;&quot;DSNMPV1TrapOutputTransporterParameters.Host = $HostParameter&quot; &quot;...</default="&quot;$STREAMING_HOME/adapters/framework/bin/start.bat&quot; &quot;/&quot;adapter_config.xml&quot; &quot;&quot;&quot;DSNMPV1TrapOutputTransporterParameters.Host = $HostParameter&quot; &quot;...&quot; />
```

## Related Information

- Adapter Toolkit: Sample cnxml File for Output Adapters [page 139]
- Adapter Integration Framework [page 157]
- Starting an Adapter [page 131]
- Stopping an Adapter [page 133]

### 2.16 Adapter Toolkit: Sample cnxml File for Output Adapters

**Adapter type:** toolkit_output. Sample cnxml file for an output adapter created using the adapter toolkit. You can use this file as reference for creating your own cnxml file for your custom adapter.

This file is located in the `$STREAMING_HOME/lib/adapters` directory. The cnxml files are required if you want to manage your custom external adapter using the smart data streaming Server. Set these properties in the studio adapter properties dialog.

<table>
<thead>
<tr>
<th>Property Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapter Configuration File</td>
<td>Property ID: configFilePath, Type: filename, (Required) Specify the path to the adapter configuration file.</td>
</tr>
</tbody>
</table>

If you use the ATTACH ADAPTER statement to attach an adapter, supply the adapter type. If you use the ATTACH ADAPTER statement with the PROPERTIES clause to override parameter values in the adapter configuration file, add the following to the adapter cnxml file:

Include the `PropertySet` parameter.

```xml
<Parameter id="propertyset"
  label="propertyset"
  descr="to look up properties in project configuration"
```
Pass the parameter-value pair as Java System Property format.

**Unix:**

```xml
<Internal id="x_unixCmdExec"
    label="Execute Command"
    type="string"
    default=""$STREAMING_HOME/adapters/framework/bin/start.sh" \
        $HostParameter" ...
/>```

**Windows:**

```xml
<Internal id="x_winCmdExec "
    label="Execute Command"
    type="string"
    default=""$STREAMING_HOME/adapters/framework/bin/start.bat" \
        $HostParameter" ...
/>```

**Related Information**

- Adapter Toolkit: Sample cnxml File for Input Adapters [page 138]
- Adapter Integration Framework [page 157]
- Starting an Adapter [page 131]
- Stopping an Adapter [page 133]

## 2.17 Debugging a Custom Adapter

Debug a custom Java adapter (that was built using the adapter toolkit) by starting it in debug mode and using an Integrated Development Environment (IDE) that supports remote debugging, such as Eclipse.

**Prerequisites**

- Install Java Runtime Environment (JRE) version 7 or higher.
- If debugging the adapter using Eclipse, install Eclipse version 3.7 (Indigo) or higher.
Context

These steps describe how to debug a custom or an example adapter using an Eclipse IDE. You can use similar steps to debug a custom adapter using another IDE that supports remote debugging.

Procedure

1. Choose an example adapter from $STREAMING_HOME/adapters/framework/examples or another adapter of your choice.
2. Compile the transporter and formatter module Java files to .class files (debug version) and compress them to .jar files using the jar.exe tool.
3. Copy these debug files to the $STREAMING_CUSTOM_ADAPTERS_HOME/lib directory.
4. Back up the custommodulesdefine.xml file in the $STREAMING_CUSTOM_ADAPTERS_HOME/config directory.
5. (Perform only if debugging an example adapter) Prepare the adapter configuration file:
   a. Copy the $STREAMING_HOME/adapters/framework/examples/custommodulesdefine.xml file to the $STREAMING_CUSTOM_ADAPTERS_HOME/config directory.
   b. Edit the set_example_env.bat (Windows) or set_example_env.sh (Unix) files and the adapter configuration file for the example adapter. Specify values for the username and password elements in the example adapter configuration file.
6. Deploy the smart data streaming project on the cluster:
   For Windows:
   ```
   %STREAMING_HOME%\bin\streamingclusteradmin --uri=esps://<host>:3<instance-number>26 --username=sybase --password=sybase --add_project --workspace-name=<workspace-name> --project-name=<project-name> --ccx=<model-name>.ccx
   ```
   For Unix:
   ```
   $STREAMING_HOME/bin/streamingclusteradmin --uri=esps://<host>:3<instance-number>26 --username=sybase --password=sybase --add_project --workspace-name=<workspace-name> --project-name=<project-name> --ccx=<model-name>.ccx
   ```
7. Start the deployed project on the cluster:
   For Windows:
   ```
   %STREAMING_HOME%\bin\streamingclusteradmin --uri=esps://<host>:3<instance-number>26 --username=sybase --password=sybase --start_project --workspace-name=<workspace-name> --project-name=<project-name>
   ```
   For Unix:
   ```
   $STREAMING_HOME/bin/streamingclusteradmin --uri=esps://<host>:3<instance-number>26 --username=sybase --password=sybase --start_project --workspace-name=<workspace-name> --project-name=<project-name>
   ```
8. Modify the `start.bat` (Windows) or `start.sh` (Unix) script file in the `$STREAMING_HOME%/adapters/framework/bin` directory to set the `suspend debug` parameter to `y`. For example:

```
set DEBUG_PARA=-Xdebug
    -Xrunjdwp:transport=dt_socket,address=8998,server=y,suspend=y
```

9. Start the adapter in debug mode as follows, where `<ADAPTER_EXAMPLE_CONFIG_FILE>` specifies the full path to the configuration file of the adapter you are debugging:

For Windows:
```
%STREAMING_HOME%/adapters/framework/bin/start.bat
<ADAPTER_EXAMPLE_CONFIG_FILE> -debug
```

For Unix:
```
$STREAMING_HOME/adapters/framework/bin/start.sh
<ADAPTER_EXAMPLE_CONFIG_FILE> -debug
```

10. Launch Eclipse.

11. Select Run > Debug Configurations.

12. On the left side of the Debug Configurations window, select Remote Java Application, then right-click and select New to create a connection to the adapter to debug.

13. On the right side of the Debug Configuration window, select the Connect tab:
   a. Specify the name of your adapter in the Name field.
   b. Use the Browse... button to select the smart data streaming project to which your adapter is connected.
   c. Select Standard (Socket Attach) from the dropdown menu for Connection Type.
   d. Specify `<host>` for the Host connection property.
   e. Specify 8998 for the Port connection property.
   f. Click Apply.

14. Select Run > Toggle Breakpoint to create a breakpoint at a specific line.
    Set all breakpoints before advancing to the next step.

15. Click Debug.

Related Information

Accessing Adapter Toolkit API Reference Information [page 19]
Create a Custom Adapter [page 18]
Formatter Modules [page 59]
Transporter Modules [page 20]
EspConnector Modules [page 108]
2.18 Statistics

An adapter constructed using the SAP HANA smart data streaming adapter toolkit maintains a set of common statistics to show its status and track its loading activities.

To get these statistics, enable the time-granularity option in the project configuration (.ccr) file, then subscribe to the _ESP_Adapter_Statistics metadata stream. For more information, see Monitoring Smart Data Streaming > Monitoring with Metadata Streams in the SAP HANA Smart Data Streaming: Configuration and Administration Guide.

All Adapters

- LastMinuteAvgRowsPerSec – average rows processed per second over the last minute
- LastHourAvgRowsPerSec – average rows processed per second over the last hour
- AdapterRunningTime – how long the adapter has been running continuously

Input Adapters Only

- TotalInputRowsNumber – the number of input rows processed
- SuccessInputRowsNumber – the number of input rows that were processed successfully
- ErrorInputRowsNumber – the number of input rows that failed
- InputLatency – average latency of all input rows processed

Output Adapters Only

- TotalOutputRowsNumber – the number of output rows processed
- SuccessOutputRowsNumber – the number of output rows that were processed successfully
- ErrorOutputRowsNumber – the number of output rows that failed
- OutputLatency – average latency of all output rows processed

Related Information

SAP HANA Smart Data Streaming: Configuration and Administration Guide
3  Creating Custom External Adapters Using SDKs

Follow general guidelines to create a custom external adapter using one of the SDKs (Java, C/C++, or .NET).

Procedure

1. Obtain an SDK instance.
2. Create credentials for the required authentication type.
3. Connect to a project for smart data streaming using these credentials.
4. Create a publisher to publish data to the smart data streaming server.
5. Create a subscriber to subscribe to records from a project in the smart data streaming server.
6. Publish or subscribe to data in smart data streaming.

In this section:

Java External Adapters [page 144]
Use the Java SDK to build a custom Java external adapter.

C/C++ External Adapters [page 149]
Use the C/C++ SDK to build custom C/C++ external adapters.

.NET External Adapters [page 152]
Use the .NET SDK to build a custom .NET external adapter.

3.1  Java External Adapters

Use the Java SDK to build a custom Java external adapter.

In this section:

Connecting to a Project [page 145]
Connect to a project using your authentication credentials.

Creating a Publisher [page 145]
Create and connect to a publisher, then publish a message.

Sample Java Code for addRow [page 146]
The addRow operation adds a single record row to messages that are published to the smart data streaming server.

Subscribing Using Callback [page 146]
Perform callbacks for new data.
We recommend direct access mode only for testing purposes.

Although you can publish in callback mode in special cases, we recommend that you do not.

### 3.1.1 Connecting to a Project

Connect to a project using your authentication credentials.

**Procedure**

1. Get the project:
   ```java
   String projectUriStr = "sap://<host>:3<instance-number>26/ws1/p1";
   Uri uri = new Uri.Builder(projectUriStr).create();
   project = sdk.getProject(uri, credentials);
   ```

2. Connect to the project:
   ```java
   project.connect(60000);
   ```
   Here, 60000 refers to the time, in milliseconds, that the smart data streaming server waits for the connection call to complete before timing out.

### 3.1.2 Creating a Publisher

Create and connect to a publisher, then publish a message.

**Procedure**

1. Create and connect to a publisher:
   ```java
   Publisher pub = project.createPublisher();
   pub.connect();
   ```

2. To create and publish a message:
   - Call a stream and the stream name
   - Call the message writer
   - Call the row writer
   - Publish
     ```java
     String streamName = "Stream1";
     ```
Stream stream = project.getStream(streamName);
MessageWriter mw = pub.getMessageWriter(streamName);
RelativeRowWriter writer = mw.getRelativeRowWriter();
mw.startEnvelope(0); // can also be mw.startTransaction() for transactions.
for (int i = 0; i < recordsToPublish.length; i++) {
    addRow(writer, incomingRecords[i], stream);
}
mw.endBlock();
pub.publish(mw);

3.1.3 Sample Java Code for addRow

The addRow operation adds a single record row to messages that are published to the smart data streaming server.

Use opcodes to update the table with a new row:

Schema schema = stream.getEffectiveSchema();
DataType[] colTypes = schema.getColumnTypes();
rowWriter.startRow();
rowWriter.setOperation(Stream.Operation.UPSERT);
for (int fieldIndex = 0; fieldIndex < schema.getColumnCount(); fieldIndex++) {
    String name = (String) colNames[fieldIndex];
    attValue = record.get(fieldIndex);
    switch(dataType){
    case BOOLEAN:     writer.setBoolean((Boolean) attValue); break;
    case INTEGER:     writer.setInteger((Integer) attValue); break;
    case MSDATE:     writer.setMSDate((Date) attValue); break;
    }//switch
} //for loop
rowWriter.endRow();

3.1.4 Subscribing Using Callback

Perform callbacks for new data.

Procedure

1. Create the subscriber options:

SubscriberOptions.Builder builder = new SubscriberOptions.Builder();
builder.setAccessMode(AccessMode.CALLBACK);
builder.setPulseInterval(pulseInterval);
SubscriberOptions opts = builder.create();

Set the access mode to CALLBACK and the pulse interval for how often you want to make the callback.

2. Create the subscriber and register the callback:

Subscriber sub = project.createSubscriber(opts);
sub.setCallback(EnumSet.allOf(SubscriberEvent.Type.class), this);
sub.subscribeStream(streamName);
sub.connect();

sub.setCallback is the class that implements the processEvent method and gets called by the callback mechanism.

3. Create the callback class, which registers with the subscriber:
   a. Implement Callback<SubscriberEvent>:
   b. Implement the getName() and processEvent(SubscriberEvent) methods.

   ```java
   public void processEvent(SubscriberEvent event) {
       switch (event.getType()) {
           case SYNC_START: dataFromLogstore=true; break;
           case SYNC_END: dataFromLogstore=false; break;
           case ERROR: handleError(event); break;
           case DATA: handleData(event); break;
           case DISCONNECTED: cleanupExit(); break;
       }
   }
   ```

A separate method named handleData is declared in this example, which is referenced in Step 4. The name of the method can vary.

i Note
When the event is received, the callback mechanism calls processEvent and passes the event to it.

4. (Optional) Use handleData to complete a separate method to retrieve and use subscribed data. Otherwise, data can be directly processed in processEvent:

   ```java
   public void handleData(SubscriberEvent event) {
       MessageReader reader = event.getMessageReader();
       String streamName = event.getStream().getName();
       while (reader.hasNextRow()) {
           RowReader row = reader.nextRowReader();
           int ops = row.getOperation().code();
           String[] colNames = row.getSchema().getColumnNames();
           List record = new ArrayList<Object>();
           for (int j = 0; j < colNames.length; ++j) {
               if (row.isNull(j)) { record.add(j, null); index++;
               continue; }
               switch (row.getSchema().getColumnTypes()[j]) {
               case BOOLEAN: record.add(j, row.getBoolean(j)); break;
               case INTEGER: record.add(j, row.getInteger(j)); break;
               case MSDATE: record.add(j, row.getMSDate(j)); break;
               } // switch
           } // for loop
           sendRecordToExternalDataSource(record);
       } // while loop
   } // handleData
   ```

The handleData event contains a message reader, gets the stream name, and uses the row reader to search for new rows as long as there is data being subscribed to. Datatypes are specified.
3.1.5 Subscribe Using Direct Access Mode

We recommend direct access mode only for testing purposes.

```java
Subscriber sub = p.createSubscriber(); sub.connect();
sub.subscribeStream("stream1");
while (true) {
    SubscriberEvent event = sub.getNextEvent();
    handleEvent(event);
}
```

3.1.6 Publish Using Callback

Although you can publish in callback mode in special cases, we recommend that you do not.

```java
PublisherOptions.Builder builder = new PublisherOptions.Builder();
builder.setAccessMode(AccessMode.CALLBACK);
builder.setPulseInterval(pulseInterval);
PublisherOptions opts = builder.create();
    Publisher pub = project.createPublisher(opts);
    pub.setCallback(EnumSet.allOf(PublisherEvent.Type.class), new
PublisherHandler(project));
    pub.connect();
```

PublisherHandler implements Callback<PublisherEvent>. It also implements the two methods: getName() and processEvent(PublisherEvent event).

The script for implementing processEvent should look like this:

```java
public void processEvent(PublisherEvent event) {
    switch (event.getType()) {
        case CONNECTED: mwriter = event.getPublisher().getMessageWriter(mstr);
                        rowwriter = mwriter.getRelativeRowWriter(); break;
        case READY: mwriter.startTransaction(0);
                    for (int j = 0; j < 100; ++j) {
                        mrowwriter.startRow();
                        mrowwriter.setOperation(Operation.INSERT);
                        for (int i = 0; i < mschema.getColumnCount(); ++i) {
                            switch (mtypes[i]) {
                                case INTEGER: mrowwriter.setInteger(int_value++);break;
                                case DOUBLE: mrowwriter.setDouble(double_value+=1.0); break;
                            }
                        }
                        mrowwriter.endRow();
                    }
        case ERROR: break;
        case DISCONNECTD:break;
    }
}
```
3.2 C/C++ External Adapters

Use the C/C++ SDK to build custom C/C++ external adapters.

In this section:

- **Getting a Project** [page 149]
  Create your authentication credentials, and use them to create a project.

- **Publishing and Subscribing** [page 150]
  Create a publisher and subscriber, and implement a callback instance.

- **handleData** [page 151]
  Sample C/C++ code for the handleData method.

### 3.2.1 Getting a Project

Create your authentication credentials, and use them to create a project.

**Context**

All calls to SDK are available as external C calls.

**Procedure**

1. Create a credentials object for authentication:

   ```
   #include <sdk/esp_sdk.h>
   #include <sdk/esp_credentials.h>
   EspError* error = esp_error_create();
   esp_sdk_start(error);
   EspCredentials* m_creds = esp_credentials_create(ESP_CREDENTIALS_USER_PASSWORD, error);
   esp_credentials_set_user(espuser.c_str(), error);
   esp_credentials_set_password(m_creds, esppass.c_str(), error);
   ```

2. Create a project:

   ```
   EspUri* m_espUri = NULL; EspProject* m_project = NULL;
   if (isCluster){
     m_espUri = esp_uri_create_string(project_uri.c_str(), error);
     m_project = esp_project_get(m_espUri, m_creds, NULL, error);
     esp_project_connect (m_project, error);
   ```
3.2.2 Publishing and Subscribing

Create a publisher and subscriber, and implement a callback instance.

Procedure

1. Create the publisher:

```c
EspPublisherOptions* publisherOptions = esp_publisher_options_create (error);
Int rc
EspPublisher * m_publisher = esp_project_create_publisher
(m_project,publisherOptions,error);
EspStream* m_stream = esp_project_get_stream (m_project,m_opts->target.c_str(),error);
rc = esp_publisher_connect (m_publisher,error);
```

2. Publish (this sample code includes syntax for adding rows to messages):

```c
EspMessageWriter* m_msgwriter   =   esp_publisher_get_writer
(m_publisher,m_stream,error);
EspRelativeRowWriter* m_rowwriter =
esp_message_writer_get_relative_rowwriter(m_msgwriter, error);
const EspSchema* m_schema = esp_stream_get_schema (m_stream,error);
int numColumns;
rc = esp_schema_get_numcolumns (m_schema, &numColumns,error);
rc = esp_message_writer_start_envelope(m_msgwriter, 0, error);
rc = esp_relative_rowwriter_start_row(m_rowwriter, error);
rc = esp_relative_rowwriter_set_operation(m_rowwriter, (const
ESP OPERATION_T)opcode, error);
int32_t colType;
for (int j = 0;j < numColumns;j++){
rc = esp_schema_get_column_type (m_schema,j,&colType,error);
switch (Type){
  case ESP DATATYPE_INTEGER:
    memcpy (&integer_val,(int32_t *)(dataValue),sizeof(uint32_t));
    rc = esp_relative_rowwriter_set_integer(m_rowwriter, integer_val,
error);
    break;
  case ESP DATATYPE_LONG:
    memcpy (&long_val,(int64_t *)(dataValue),sizeof(int64_t));
    rc = esp_relative_rowwriter_set_long(m_rowwriter, long_val,
error);
    break;
}
rc = esp_relative_rowwriter_end_row(m_rowwriter, error);
rc = esp_message_writer_end_block(m_msgwriter, error);
rc = esp_publisher_publish(m_publisher, m_msgwriter, error);
```

3. Create the subscriber options:

```c
EspSubscriberOptions * m_subscriberOptions = esp_subscriber_options_create
(error);
int rc = esp_subscriber_options_set_access_mode(options, CALLBACK_ACCESS,
_m_error);
EspSubscriber * m_subscriber = esp_project_create_subscriber
(m_project,m_subscriberOptions,error);
rc = esp_subscriber_options_free(options, m_error);
rc = esp_subscriber_set_callback(subscriber, ESP_SUBSCRIBER_EVENT_ALL,
subscriber_callback, NULL, m_error);
```
subscriber_callback is global function that will get called up.

4. Subscribe using callback:

```c
void subscriber_callback(const EspSubscriberEvent * event, void * data) {
    uint32_t type;
    rc = esp_subscriber_event_get_type(event, &type, error);
    switch (type) {
        case ESP_SUBSCRIBER_EVENT_CONNECTED:
            init(event, error); break;
        case ESP_SUBSCRIBER_EVENT_SYNC_START:
            fromLogStore = true; break;
        case ESP_SUBSCRIBER_EVENT_SYNC_END:
            fromLogStore = false; break;
        case ESP_SUBSCRIBER_EVENT_DATA:
            handleData(event, error); break;
        case ESP_SUBSCRIBER_EVENT_DISCONNECTED:
            cleanupaExit(); break;
        case ESP_SUBSCRIBER_EVENT_ERROR:
            handleError(event, error); break;
    }
}
```

3.2.3 handleData

Sample C/C++ code for the handleData method.

```c
EspMessageReader * reader = esp_subscriber_event_get_reader(event, error);
EspStream * stream = esp_message_reader_get_stream(reader, error);
const EspSchema * schema = esp_stream_get_schema(stream, error);
EspRowReader * row_reader;
int32_t int_value; int64_t long_value; time_t date_value; double double_value;
int numcolumns, numrows, type;
rc = esp_schema_get_numcolumns(schema, &numcolumns, error);
while ((row_reader = esp_message_reader_next_row(reader, error)) != NULL) {
    for (int i = 0; i < numcolumns; ++i) {
        rc = esp_schema_get_column_type(schema, i, &type, error);
        switch (type) {
            case ESP_DATATYPE_INTEGER:
                rc = esp_row_reader_get_integer(row_reader, i, &int_value, error);
                break;
            case ESP_DATATYPE_LONG:
                rc = esp_row_reader_get_long(row_reader, i, &long_value, error);
                break;
            case ESP_DATATYPE_DATE:
                rc = esp_row_reader_get_date(row_reader, i, &date_value, error);
                break;
        }
    }
}
```
3.3 .NET External Adapters

Use the .NET SDK to build a custom .NET external adapter.

In this section:

- Connecting to the Smart Data Streaming Server [page 152]
  Set credentials and .NET server options when you connect to the smart data streaming server.
- Connecting to a Project [page 153]
  Use sample .NET code to connect to a project.
- Publishing [page 153]
  Create a publisher, add rows, and complete the publishing process.
- Connecting to a Subscriber [page 154]
  Create and connect to a new subscriber.
- Subscribing Using Callback Mode [page 154]
  Perform callbacks for new data.

3.3.1 Connecting to the Smart Data Streaming Server

Set credentials and .NET server options when you connect to the smart data streaming server.

Procedure

1. Create an error message store for setting credentials and .NET server options:
   ```csharp
   NetEspError error = new NetEspError();
   ```

2. Set a new URI:
   ```csharp
   NetEspUri uri = new NetEspUri();
   uri.set_uri("esps://cepsun64amd.mycompany.com:3<instance-number>26", error);
   ```

3. Create your credentials:
   ```csharp
   NetEspCredentials creds = new NetEspCredentials(NetEspCredentials.NET_ESP_CREDENTIALS_T.NET_ESP_CREDENTIALS_SERVER_RSA);
   creds.set_user("pengg");
   creds.set_password("1234");
   creds.set_keyfile("..\test_data\keys\client.pem");
   ```

4. Set options:
   ```csharp
   NetEspServerOptions options = new NetEspServerOptions();
   options.set_mode(NetEspServerOptions.NET_ESP_ACCESS_MODE_T.NET_CALLBACK_ACCESS);
   server = new NetEspServer(uri, creds, options);
   int rc = server.connect(error);
   ```
3.3.2 Connecting to a Project

Use sample .NET code to connect to a project.

Procedure

1. Get the project:
   ```
   NetEspProject project = server.get_project("test", "test", error);
   ```

2. Connect to the project:
   ```
   project.connect(error);
   ```

3.3.3 Publishing

Create a publisher, add rows, and complete the publishing process.

Procedure

1. Create a publisher:
   ```
   NetEspPublisher publisher = project.create_publisher(null, error);
   ```

2. Connect to the publisher:
   ```
   Publisher.connect(error);
   ```

3. Get a stream:
   ```
   NetEspStream stream = project.get_stream("WIN2", error);
   ```

4. Get the message writer:
   ```
   NetEspMessageWriter writer = publisher.get_message_writer(stream, error);
   ```

5. Get and start the row writer, and set an opcode to insert one row:
   ```
   NetEspRelativeRowWriter rowwriter = writer.get_relative_row_writer(error);
   rowwriter.start_row(error);
   rowwriter.set_opcode(1, error);
   ```
(Optional) If publishing in transaction mode, use these arguments to add multiple rows:

```java
NetEspRelativeRowWriter rowwriter = writer.get_relative_row_writer(error);
for(int i=0; i<100; i++){
    rowwriter.start_row(error);
    //add row columns' values
    rowwriter.end_row(error);
}
```

6. Publish data:

```java
rc = publisher.publish(writer, error);
```

### 3.3.4 Connecting to a Subscriber

Create and connect to a new subscriber.

**Procedure**

1. Create a subscriber:

```java
NetEspSubscriberOptions options = new NetEspSubscriberOptions();
options.set_mode(NetEspSubscriberOptions.NET_ESP_ACCESS_MODE_T.NET_CALLBACK_ACCESS);
NetEspSubscriber subscriber = new NetEspSubscriber(options, error);
```

2. Connect to the subscriber:

```java
Subscriber.connect(error);
```

### 3.3.5 Subscribing Using Callback Mode

Perform callbacks for new data.

**Procedure**

1. Set the subscriber options:

```java
NetEspSubscriberOptions options = new NetEspSubscriberOptions();
options.set_mode(NetEspSubscriberOptions.NET_ESP_ACCESS_MODE_T.NET_CALLBACK_ACCESS);
NetEspSubscriber subscriber = new NetEspSubscriber(options, error);
```
2. Create the callback instance:

```java
NetEspSubscriber.SUBSCRIBER_EVENT_CALLBACK callbackInstance = new
NetEspSubscriber.SUBSCRIBER_EVENT_CALLBACK(subscriber_callback);
```

3. Create the callback registry:

```java
subscriber.set_callback(NetEspSubscriber.NET_ESP_SUBSCRIBER_EVENT.NET_ESP_SUBSCRIBER_EVENT_ALL, callbackInstance, null, error);
```

4. Connect to the subscriber:

```java
subscriber.connect(error);
```

5. Subscribe to a stream:

```java
subscriber.subscribe_stream(stream, error);
```

6. Implement the callback:

```java
Public static void subscriber_callback(NetEspSubscriberEvent event, ValueType data) {
    switch (evt.getType()) {
        case (uint) (NetEspSubscriber.NET_ESP_SUBSCRIBER_EVENT.NET_ESP_SUBSCRIBER_EVENT_CONNECTED):
            Console.WriteLine("the callback happened: connected!");
            break;
        (uint) (NetEspSubscriber.NET_ESP_SUBSCRIBER_EVENT.NET_ESP_SUBSCRIBER_EVENT_DATA):
    }
}
```

7. (Optional) Use handleData to complete a separate method to retrieve and use subscribed data.

```java
NetEspRowReader row_reader = null;
while ((row_reader = evt.getMessageReader().next_row(error)) != null) {
   for (int i = 0; i < schema.get_numcolumns(); ++i) {
      if (row_reader.is_null(i) == 1) {
         Console.Write("null, ");
         continue;
      }
      switch (NetEspStream.getType(schema.get_column_type((uint)i, error))) {
         case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_INTEGER:
            ivalue = row_reader.get_integer(i, error);
            Console.Write(ivalue + ", ");
            break;
         case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_LONG:
            lvalue = row_reader.get_long(i, error);
            Console.Write(lvalue + ", ");
            break;
         case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_FLOAT:
            fvalue = row_reader.get_float(i, error);
            Console.Write(fvalue + ", ");
            break;
         case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_STRING:
            svalue = row_reader.get_string(i, error);
            Console.Write(svalue);
            break;
      }
   }
}
```
case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_SECONDDATE:
    dvalue = row_reader.get_seconddate(i, error);
    Console.Write(dvalue + ", ");
    break;

case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_MSDATE:
    tvalue = row_reader.get_msdate(i, error);
    Console.Write(tvalue + ", ");
    break;

case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_BOOLEAN:
    boolvalue = row_reader.get_boolean(i, error);
    Console.Write(boolvalue + ", ");
    break;

case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_BINARY:
    uint buffersize = 256;
    binvalue = row_reader.get_binary(i, buffersize, error);
    break;

case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_INTERVAL:
    intervalvalue = row_reader.get_interval(i, error);
    Console.Write(intervalvalue + ", ");
    break;

case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_MONEY01:
    mon = row_reader.get_money(i, error);
    Console.Write(mon.get_long(error) + ", ");
    break;

case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_MONEY02:
    lvalue = row_reader.get_money_as_long(i, error);
    Console.Write(lvalue + ", ");
    break;

case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_MONEY03:
    mon = row_reader.get_money(i, error);
    Console.Write(mon.get_long(error) + ", ");
    break;

case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_MONEY10:
    mon = row_reader.get_money(i, error);
    Console.Write(mon.get_long(error) + ", ");
    break;

case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_MONEY15:
    mon = row_reader.get_money(i, error);
    Console.Write(mon.get_long(error) + ", ");
    break;

case NetEspStream.NET_DATA_TYPE_T.NET_ESP_DATATYPE_BIGDATETIME:
    bdt2 = row_reader.get_bigdatetime(i, error);
    long usecs = bdt2.get_microseconds(error);
    Console.Write(usecs + ", ");
    break;
}
}

8. Disconnect from the subscriber:

    rc = subscriber.disconnect(error);
}
4 Adapter Integration Framework

The smart data streaming adapter integration framework provides a mechanism for integrating custom external adapters with the smart data streaming server and smart data streaming plugin for SAP HANA studio.

A custom external adapter is any application that publishes or subscribes to a stream or a set of streams in smart data streaming using the publisher and subscriber APIs from any of the smart data streaming SDKs.

To start and stop external adapters together with a project for smart data streaming or visually from within studio, create an instance of the adapter plug-in by creating a specialized `cnxml` configuration file for the adapter. Within this file, specify command lines that the smart data streaming server can invoke during various stages of adapter runtime and initialization.

In this section:

cnxml Configuration File [page 157]
   The external adapter framework defines the structure of the `cnxml` file that contains the adapter properties.

Creating a cnxml File for a Custom Adapter [page 158]
   Create a `cnxml` configuration file for your custom external adapter so that you can configure the adapter in the SAP HANA Streaming Development perspective, and start and stop it with a project for smart data streaming.

Related Information

Adapter Toolkit: Sample cnxml File for Output Adapters [page 139]
Adapter Toolkit: Sample cnxml File for Input Adapters [page 138]
Starting an Adapter [page 131]
Stopping an Adapter [page 133]

4.1 cnxml Configuration File

The external adapter framework defines the structure of the `cnxml` file that contains the adapter properties.

The external adapter configuration file is an XML file that contains the properties and commands used by smart data streaming to start and stop the external adapter, and to optionally run schema discovery, as well as other information that allows you to configure the adapter from studio.

Here is the structure of a `cnxml` file:

```xml
<Adapter>
    type = Can be input or output
</Adapter>
```
external = True
id = Required to be unique to the adapter
label = The adapter's name in studio
descr = Description of adapter functionalities

.Library
  file = simple_ext (Always use this for custom external adapters)
  type = Binary
</Library>

</Special>

<Internal>
  **These values are not configurable from studio. simple_ext has a
  function that parses this cnxml file and looks up the internal fields to find
  particular commands and their default values**
  
  id = See $STREAMING_HOME/lib/adapters/simple_ext.cnxml.template for a sample
  cnxml file that contains all possible internal parameters and comment blocks
  indicating their usage.
  
  label = Description of function
  type = Datatype of function
  default = This is the command that simple_ext executes. For example, if you
  are using x_unixCmdExec, it calls streamingconvert with targets you specify
  followed by an upload.
</Internal>

</Section>

**These parameter values are visible (except for id) and configurable in
studio.**

  id = The property id that you can reference to in <Internal> command
calls with $(id's name). This is what you reference when writing an adapter in
CCL.
  label = The property name that appears in studio.
  descr = Description of adapter property.
  type = The property datatype.
  use = Whether the property is required, optional, or advanced. In
  studio, required properties appear in red and advanced ones appear on a separate
  tab.
  default = Default value to use if you do not set a value for the
  property.

---

## 4.2 Creating a cnxml File for a Custom Adapter

Create a .cnxml configuration file for your custom external adapter so that you can configure the adapter in the SAP HANA Streaming Development perspective, and start and stop it with a project for smart data streaming.

### Procedure

1. Specify attributes for the Adapter section:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>(Required) Input or output adapter.</td>
</tr>
<tr>
<td>external</td>
<td>(Required) Set to true for external adapters.</td>
</tr>
<tr>
<td>id</td>
<td>(Required) Unique identifier for your adapter. This value is listed in the type parameter within the ATTACH ADAPTER statement.</td>
</tr>
</tbody>
</table>
2. Specify attributes for the Library section:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>(Required) Name of the adapter. This name appears in the SAP HANA Streaming Development perspective if you hover over the adapter icon in the visual editor.</td>
</tr>
<tr>
<td>description</td>
<td>(Required) Purpose of the adapter. This is also visible in the SAP HANA Streaming Development perspective.</td>
</tr>
</tbody>
</table>

3. Specify internal parameters for the Special section. See the "$STREAMING_HOME/lib/adapters/simple_ext.cnxml.template" sample cnxml file for a complete list of internal parameters and usage details.

**Note**
Any commands specified in the cnxml file for an adapter cannot contain the following strings: "mv", "rm", or "del".

4. Specify adapter parameters for the Section section. These parameters are visible and configurable in studio.

For each parameter, specify:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>(Required) The property ID that you can reference in &lt;Internal&gt; command calls with $&lt;ID name&gt;. This is what you reference when specifying adapter properties for an adapter in CCL.</td>
</tr>
<tr>
<td>label</td>
<td>(Required) The property name that appears in studio.</td>
</tr>
<tr>
<td>descr</td>
<td>(Required) A description of the adapter property.</td>
</tr>
<tr>
<td>type</td>
<td>(Required) The property datatype.</td>
</tr>
<tr>
<td>use</td>
<td>(Required) Whether the property is required, optional, or advanced. In studio, required properties appear in red and advanced ones appear on a separate tab.</td>
</tr>
<tr>
<td>default</td>
<td>(Optional) The default value to use if you do not set a value for the property.</td>
</tr>
</tbody>
</table>

In this section:

**Example cnxml Configuration File [page 160]**
Example of a cnxml configuration file that uses some of the utilities shipped with smart data streaming (streamingconvert, streamingupload, streamingprojectclient, and streamingdiscxmlfiles) to fully define a functional external adapter that supports file directory browsing, source stream creation, and data loading.
External Adapter Properties [page 162]

See $STREAMING_HOME/lib/adapters/simple_ext.cnxml.template for a sample cnxml file that you can copy and customize. It includes all possible internal parameters, along with comment blocks that include usage notes for each parameter.

External Adapter Commands [page 163]

External adapter commands fall into two categories: those that run on the same host as studio, and those that run on the same host as the server.

User-Defined Parameters and Parameter Substitution [page 165]

You can create internal parameters and any number of user-defined parameters in the cnxml file.

Autogenerated Parameter Files [page 166]

The basic external adapter framework, when started, writes its set of parameters (system and user-defined) to a parameter file.

configFilename Parameter [page 167]

The configFilename parameter enables you to specify user-editable configuration files in the studio.

Task overview: Create a Custom Adapter [page 18]

Previous task: Configuring a New Adapter [page 118]

Next task: Starting an Adapter [page 131]

4.2.1 Example cnxml Configuration File

Example of a cnxml configuration file that uses some of the utilities shipped with smart data streaming (streamingconvert, streamingupload, streamingprojectclient, and streamingdiscxmlfiles) to fully define a functional external adapter that supports file directory browsing, source stream creation, and data loading.

In this example, long lines of script are split for readability and to avoid formatting issues. If you are using this example to create your own external adapter configuration file, ensure that all command properties are on a single line, regardless of length:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Adapter type="input" external="true"
    id="simplified_xml_input_plugin"
    label="Simplified external XML file input plugin Adapter"
    descr="Example of uploading an XML file through a simple external Adapter" />

<Library file="simple_ext" type="binary"/>

<!--
The special section contains the special internal parameters that are prefixed with "x_". Although these are parameters, the framework requires that you define these using the <Internal .../> element. They are hidden within studio.
-->

<Special>
    <Internal id="x_initialOnly"
        label="Does Initial Loading Only"
        descr="Do initial loading, or the continuous loading"
        type="boolean"
```
default="true"
/>
</Internal id="x_addParamFile"
label="Add Parameter File"
type="boolean"
default="false"
/>
</Internal id="x_killRetryPeriod"
label="Period to repeat the stop command until the process exits"
type="int"
default="1"
/>
<!--
Convert a file of xml record to smart data streaming binary format using
streamingconvert;
pipe into the streamingupload program, naming the upload connection:

$platformStream.$platformConnection
-->
</Internal id="x_unixCmdExec"
label="Execute Command"
type="string"
default="$STREAMING_HOME/bin/streamingconvert -p $platformCommandPort
&amp;quot;$directory/$filename&amp;quot; | $STREAMING_HOME/bin/streamingupload -m
$platformStream.$platformConnection -p $platformCommandPort"
/>
</Internal id="x_winCmdExec"
label="Execute Command"
type="string"
default="$+/{$STREAMING_HOME/bin/streamingconvert} -p $platformCommandPort
&amp;quot;$directory/$filename&amp;quot; | $+/{$STREAMING_HOME/bin/streamingupload} -m
$platformStream.$platformConnection -p $platformCommandPort"
/>
<!--
use the streamingprojectclient command to stop an existing
streamingupload connection named:

$platformStream.$platformConnection
-->
</Internal id="x_unixCmdStop"
label="Stop Command"
type="string"
default="$STREAMING_HOME/bin/streamingprojectclient -p
$platformCommandPort 'kill every {$platformStream.$platformConnection}' &lt;/dev/null"
/>
</Internal id="x_winCmdStop"
label="Stop Command"
type="string"
default="$+/{$STREAMING_HOME/bin/streamingprojectclient} -p
$platformCommandPort &quot;kill every {$platformStream.$platformConnection}&quot; &lt;/null"
/>
<!--
Use the streamingdiscxmlfiles command to do schema discovery.
The command below will have '-o "&lt;temp file&gt;"' added to it. It
writes the discovered schema in this file.
-->
</Internal id="x_unixCmdDisc"
label="Discovery Command"
type="string"
default="$STREAMING_HOME/bin/streamingdiscxmlfiles -d &quot;$directory&quot;"
/>
</Internal id="x_winCmdDisc"
label="Discovery Command"
type="string"
default="$+/{$STREAMING_HOME/bin/streamingdiscxmlfiles} -d &quot;$directory"+$*/
{$directory}&quot;"/>
4.2.2 External Adapter Properties

See $STREAMING_HOME/lib/adapters/simple_ext.cnxml.template for a sample cnxml file that you can copy and customize. It includes all possible internal parameters, along with comment blocks that include usage notes for each parameter.

<table>
<thead>
<tr>
<th>Property ID</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_addParamFile</td>
<td>boolean</td>
<td>Determines if the parameter file name is automatically appended to all x_cmd* strings. For example, if you specify the command as cmd -f, and this parameter is set to true, the actual command is executed as cmd -f &lt;value-of-x_paramFile&gt;.</td>
</tr>
<tr>
<td>x_initialOnly</td>
<td>boolean</td>
<td>If true, performs only initial loading. Set to false for continuous loading. Initial loading is useful for adapters that start, load some static data and then finish; therefore, allowing another adapter group to start up in a staged loading scenario.</td>
</tr>
<tr>
<td>x_killRetryPeriod</td>
<td>integer</td>
<td>If this parameter is greater than 0, the x_{unix,win}CmdStop command is retried every x_killRetry seconds, until the framework detects that the x_{unix,win}CmdExec command has returned. If the parameter is zero, run the x_{unix,win}CmdStop command once and assume that it has stopped the x_{unix,win}CmdExec command.</td>
</tr>
<tr>
<td>Property ID</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| x_paramFile | string | Specifies the file name of the adapter framework, which writes all internal and user-defined parameters. It may use other internal parameters to specify the file name. For example, 
/tmp/mymodel.$platformStream. $platformConnection. $platformCommandPort.cfg |
| x_paramFormat | string | Stops the adapter, and runs in a separate thread, which returns and stops processes created with x_{unix,win}CmdExec. |
| x_unixCmdClean/ x_winCmdClean | string | Specifies the command executed to perform cleanup after the adapter stops (x_{unix,win}CmdExec returns). |
| x_unixCmdConfig/ x_winCmdConfig | string | Specifies the command that preprocesses (for example, parsing and checking parameters) the parameter file before the adapter starts, such as parsing and checking parameters. This command might convert the parameters to real format (by reading, parsing, and rewriting the parameter file), which is required by the execution command. If the configure command fails (non-zero return), the adapter fails to start. |
| x_unixCmdDisc/ x_winCmdDisc | string | Specifies the command that performs discovery. This command adds the -o "<temporary-disc-file-name>" argument, which writes the discovery XML to a file, before it executes. |
| x_unixCmdExec/ x_winCmdExec | string | Specifies the command that starts the adapter. When the command returns, it indicates that the adapter is done running. |
| x_unixCmdStop/ x_winCmdStop | string | Specifies the command that stops the adapter. The stop command runs from a separate thread and stops all processes created with the x_{unix,win}CmdExec command, which returns the x_{unix,win}CmdExec command. |

**Note**

Some of the commands have both Windows or Unix formats because their paths, environment variables, and script files differ. At runtime, SAP HANA smart data streaming determines which command to use based on the operating system it is running on. All commands have substituted parameters and environment variables. When a command is passed to shell or cmd.exe, it runs its own round of substitution.

### 4.2.3 External Adapter Commands

External adapter commands fall into two categories: those that run on the same host as studio, and those that run on the same host as the server.

Any commands specified in the cnxml file for an adapter cannot contain the following strings: "mv", "rm", or "del".

The discovery commands, x_unixDiscCmd and x_winDiscCmd, always run on the studio host. All other commands run on the server host.
The studio and server are frequently run on the same host, so the development of all command and driving scripts for the custom adapter are straightforward. The configuration becomes more complex during remote execution when studio and the server are running on different hosts.

For example, if the studio is running on a Windows host, and the server is set up through studio to execute on a remote Linux host, this implies that both the discovery command and the discovery file name that the framework generates are running, and they are generated in a Windows environment. The path to the discovery file is a Windows-specific path with drive letters and forward-slash (‘/’) characters used as path separators. In this case, the developer of the connector should write the discovery command to run in a Windows environment while coding all other commands to remotely execute on the Linux box using a user-configured ssh or rsh command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_unixCmdConfig</td>
<td>The configure command should perform any required parsing and parameter checking. It can also convert the parameters into the real format expected by the execution command by reading, parsing, and rewriting the parameter file. If the configure command fails (non-zero return), it is reported as a reset() error, and the adapter fails to start.</td>
</tr>
<tr>
<td>x_winCmdConfig</td>
<td></td>
</tr>
<tr>
<td>x_unixCmdExec</td>
<td>When the server starts the adapter, it executes this command with its ending indicating that the connector has finished.</td>
</tr>
<tr>
<td>x_winCmdExec</td>
<td></td>
</tr>
<tr>
<td>x_unixCmdStop</td>
<td>The stop command runs from a separate thread; it should stop all processes created with the x_{unix,win}CmdExec command, thus causing the x_{unix,win}CmdExec to return.</td>
</tr>
<tr>
<td>x_winCmdStop</td>
<td></td>
</tr>
<tr>
<td>x_unixCmdClean</td>
<td>The clean command runs after the server has stopped the connection, that is, when x_{unix,win}CmdExec returns.</td>
</tr>
<tr>
<td>x_winCmdClean</td>
<td></td>
</tr>
<tr>
<td>x_winDiscCmd</td>
<td>This command is for schema discovery. It should write a discovery file into the file name passed to it. Append the parameter -o &lt;temporary disc filename&gt; command before executing it.</td>
</tr>
</tbody>
</table>

```
<discover>
  <table name="table_name_1" />
  <column name="col_name_1" type="col_type_1"/>
    ...
  <column name="col_name_k" type="col_type_k"/>
  </table>
  ...
  ...
  <table name="table_name_n" />
  <column name="col_name_1" type="col_type_1"/>
    ...
  <column name="col_name_1" type="col_type_1"/>
  </table>
</discover>
```
4.2.4 User-Defined Parameters and Parameter Substitution

You can create internal parameters and any number of user-defined parameters in the cnxml file.

All system and user-defined parameters can be referenced in the command or script arguments. These parameters behave in a similar way to shell substitution variables. This simple example includes long lines that have been split for readability:

```xml
<Internal id="x_unixCmdExec"
    label="Execute Command"
    type="string"
    default="${STREAMING_HOME}/bin/streamingconvert
            -p $platformCommandPort \"$directory/\"$filename\"
            ${STREAMING_HOME}/bin/streamingupload
            -m $platformStream.$platformConnection
            -p $platformCommandPort"/>
```

External environment variables, such as STREAMING_HOME, may be expanded, as well as internal system parameters (platformCommandPort) and user-defined parameters (filename). The full semantics for parameter expansion are:

- `$name`  
- `${name}`  
- `${name=value?substitution[:substitution]}`  
- `${name<>value?substitution[:substitution]}`  
- `${name!=value?substitution[:substitution]}`  
- `${name==value?substitution[:substitution]}`  
- `${name<value?substitution[:substitution]}`  
- `${name<=value?substitution[:substitution]}`  
- `${name>value?substitution[:substitution]}`  
- `${name>=value?substitution[:substitution]}`

All forms with `{}` may have a + added after $ (for example, $+{name}). The presence of + means that the result of the substitution is parsed again, and any values in it are substituted. The \ symbol escapes the next character and prevents any special interpretation.

The conditional expression compares the value of a parameter with a constant value and uses either the first substitution on success or the second substitution on failure. The comparisons == and != try to compare the values as numbers. The = comparisons and <> try to compare values as strings. Any characters like ?, :, and } in the values must be shielded with \\. The characters { and } in the substitutions must be balanced, all unbalanced braces must be shielded with \. The quote characters are not treated as special.

This form of substitution, $+{...}, may contain references to other variables. This is implemented by passing the result of a substitution through one more round of substitution. The consequence is that extra layers of \ may be needed for shielding. For example, the string $+{name=?\\} produces one \ if the parameter name is empty. On the first pass, each pair of backslashes is turned into one backslash, and on the second pass, \ turns into a single backslash.

Special substitution syntax for Windows convenience:

<table>
<thead>
<tr>
<th>Substitution Syntax</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>${/value}</td>
<td>Replaces all the forward-slashes in the value with backslashes, for convenience of specifying the Windows paths that would otherwise require all slashes to be escaped.</td>
</tr>
<tr>
<td>$+{/value}</td>
<td></td>
</tr>
<tr>
<td>Substitution Syntax</td>
<td>Behavior</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
</tr>
<tr>
<td>$%{value}</td>
<td>Replaces all the % with %% as escaping for Windows.</td>
</tr>
<tr>
<td>$+%{value}</td>
<td></td>
</tr>
</tbody>
</table>

If the resulting string is passed to shell or cmd.exe for execution, shell or cmd.exe performs its own substitution as well.

This example uses some of the more powerful substitution features to define the execution command as in the simple example. However, you may make use of the conditional features to support optional authentication and encryption, and an optional user-defined date format:

```xml
<Internal id="x_unixCmdExec"
  label="Execute Command"
  type="string"
  default="$STREAMING_HOME/bin/streamingconvert
  $(platformSsl==1?-e)
  $+%{dateFormat<>?-m '$dateFormat'}
  -c '$%{user=?user:$user}:$password'
  -p $platformCommandPort
  "$directory/$filename" | $STREAMING_HOME/bin/streamingupload
  $(platformSsl==1?-e)
  -m $platformStream.$platformConnection
  -c 'User:$password'
  -p $platformCommandPort"
/>
```

## 4.2.5 Autogenerated Parameter Files

The basic external adapter framework, when started, writes its set of parameters (system and user-defined) to a parameter file.

This file is written in one of the following:

- Java properties
- Shell assignments
- Simple XML format

Commands then have full access to the parameter file.

This is an example of parameters in the simplified_xml_input_plugin.cnxml file:

```xml
<Internal id="x_paramFile"
  label="Parameter File"
  type="string"
  default="/tmp/PARAMETER_FILE.txt"
/>

<Internal id="x_paramFormat"
  label="Parameter Format"
  type="string"
  default="prop"
/>

<Internal id="x_addParamFile"
  label="Add Parameter File"
  type="boolean"
/>
The parameter file is written to /tmp/PARAMETER_FILE.txt or to a full list of parameters in a Java properties format. You can specify this format as shell for shell assignments, or as xml for a simple XML format.

Here is a sample parameter file:

directory=/home/sjk/work/alerei/cimarron/branches/3.1/examples/input/xml_tables
filename=trades.xml
platformAuth=none
platformCommandPort=31415
platformConnection=Connection1
platformHost=sjk-laptop
platformSqlPort=22200
platformSsl=0
platformStream=Trades

You can specify the format as shell for shell assignments, or as xml for a simple XML format.

When you specify x_addParamFile as true, as shown here, the argument /tmp/PARAMETER_FILE.txt is added to all commands before they execute:

```xml
<Internal id="x_addParamFile"
  label="Add Parameter File"
  type="boolean"
  default="true"
/>```

4.2.6 configFilename Parameter

The configFilename parameter enables you to specify user-editable configuration files in the studio.

If you create a user-defined configFilename parameter, clicking in the value portion of this field in studio produces a file selector dialog, allowing you to choose a file on the local file system. Right-clicking the read-only name shows you a different dialog, allowing you to modify file contents. This lets you specify user-editable configuration files.
5 Adapter Parameter Datatypes

Describes the datatypes, enumerated in the $STREAMING_HOME/etc/Adapter.xsd file, that may be used for parameters specified in an adapter’s cnxml file.

Some exceptions for custom external adapters are noted in the datatype descriptions.

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
</table>
| boolean     | Value is true or false. You can substitute true or false with these paired options:  
  • 0 or 1  
  • y or n  
  • on or off  
  • yes or no  
  These options are case-insensitive. |
| choice      | A list of custom values from which a user selects one value. |
| configFilename | Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but can be no more than 65535 bytes. |
| directory   | Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but can be no more than 65535 bytes. |
| double      | A 64-bit numeric floating point with double precision. The range of allowed values is approximately $-10^{308}$ through $10^{308}$. Equivalent to float. |
| float       | A 64-bit numeric floating point with double precision. The range of allowed values is approximately $-10^{308}$ through $10^{308}$. Equivalent to double. |
| filename    | Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but can be no more than 65535 bytes. |
| int         | A signed 32-bit integer value. The range of allowed values is $-2147483648$ to $+2147483647$ ($-2^{31}$ to $2^{31}-1$). Constant values that fall outside of this range are automatically processed as long datatypes.  
  To initialize a variable, parameter, or column with the lowest negative value, specify (-2...7) -1 instead to avoid CCL compiler errors. For example, specify (-2147483647) -1 to initialize a variable, parameter, or column with a value of -2147483648. |
| password    | Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but can be no more than 65535 bytes.  
  While entering a value for this field, user sees ‘*’ for every character. |
<table>
<thead>
<tr>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>permutation</td>
<td>Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but can be no more than 65535 bytes. This datatype is not supported for custom external adapters.</td>
</tr>
<tr>
<td>query</td>
<td>A string value that studio creates from the table name. This datatype is not supported for custom external adapters.</td>
</tr>
<tr>
<td>range</td>
<td>An integer value for which user can define lower and upper limits. For example,</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Parameter id=&quot;port&quot; label=&quot;KDB Port&quot; descr=&quot;IP port of the database listener&quot; type=&quot;range&quot; rangeLow=&quot;0&quot; rangeHigh=&quot;65535&quot; default=&quot;5001&quot; use=&quot;required&quot; /&gt;</code></td>
</tr>
<tr>
<td>runtimeDirectory</td>
<td>Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but no less than 65535 bytes. This datatype is not supported for custom external adapters.</td>
</tr>
<tr>
<td>runtimeFilename</td>
<td>Runtime file name, if different from discovery time file name. Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but no more than 65535 bytes. This datatype is not supported for custom external adapters.</td>
</tr>
<tr>
<td>string</td>
<td>Variable-length character string, with byte values encoded in UTF-8. Maximum string length is platform-dependent, but no more than 65535 bytes.</td>
</tr>
<tr>
<td>tables</td>
<td>List of choices returned by the <code>getTables()</code> that are defined in the adapter.</td>
</tr>
<tr>
<td>text</td>
<td>A value capable of storing multiline text. This datatype is not supported for custom external adapters.</td>
</tr>
<tr>
<td>uint</td>
<td>Positive integer value. The range of allowed values is 0 to 0xffffffff.</td>
</tr>
</tbody>
</table>

For more information about the specific datatypes that are supported by each adapter, as well as for its datatype mapping descriptions, see the section on that adapter.
6 Methods of Defining Adapter Parameters for Managed and Unmanaged Mode Configuration

Default parameters are specified using one of four different methods: the CCL ATTACH ADAPTER statement; the adapter cnxml file; the adapter_config.xml file; and in the adapter Java source code. Depending on which method is used, and the configuration mode, parameters are loaded into the adapter based on set hierarchies.

In managed mode, the adapter loads default parameters in the following order:

1. From the ATTACH ADAPTER statement. When the adapter starts up in managed mode, it loads parameters for the project from the statement.
2. If values are not defined in the ATTACH ADAPTER statement, the adapter cnxml file is used.
3. If default parameters are not defined in either the ATTACH ADAPTER statement or the adapter cnxml file, the adapter_config.xml file is used.
4. If default parameters are not defined in the ATTACH ADAPTER statement, the adapter cnxml file, or the adapter_config.xml, the adapter Java source code is used.

In unmanaged mode, the adapter loads default parameters in the following order:

1. From the adapter_config.xml file.
2. If default parameters are not defined in the adapter_config.xml file, the adapter Java source code is used.
Important Disclaimers and Legal Information

Coding Samples

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, unless damages were caused by SAP intentionally or by SAP’s gross negligence.

Accessibility

The information contained in the SAP documentation represents SAP’s current view of accessibility criteria as of the date of publication; it is in no way intended to be a binding guideline on how to ensure accessibility of software products. SAP in particular disclaims any liability in relation to this document. This disclaimer, however, does not apply in cases of willful misconduct or gross negligence of SAP. Furthermore, this document does not result in any direct or indirect contractual obligations of SAP.

Gender-Neutral Language

As far as possible, SAP documentation is gender neutral. Depending on the context, the reader is addressed directly with "you", or a gender-neutral noun (such as "sales person" or "working days") is used. If when referring to members of both sexes, however, the third-person singular cannot be avoided or a gender-neutral noun does not exist, SAP reserves the right to use the masculine form of the noun and pronoun. This is to ensure that the documentation remains comprehensible.

Internet Hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint about where to find related information. SAP does not warrant the availability and correctness of this related information or the ability of this information to serve a particular purpose. SAP shall not be liable for any damages caused by the use of related information unless damages have been caused by SAP’s gross negligence or willful misconduct. All links are categorized for transparency (see: http://help.sap.com/disclaimer).